이론

❖ Coroutine

- Coroutine 개요
- Coroutine 활용

❖ HTTP 통신

- Volley 라이브러리를 활용한 통신
- Retrofit 라이브러리를 활용한 통신

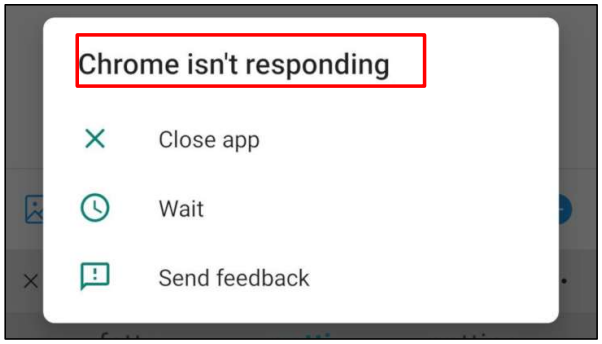
❖ 기타 편의 기능

- 파일 다운로더
- Glide 라이브러리 : 움직이는 이미지(GIF)

Coroutine 개요

❖ Coroutine 이란?

- Android의 **비동기 프로그래밍(동시성 프로그래밍) 을 작성하기 위한 기능**
- **스레드 차단 없이 루틴 수행이 가능**하며, 단일 스레드에서 **여러 Coroutine 수행** 가능
- **Jetpack 라이브러리**에 포함된 Coroutine을 활용하여 앱을 **개발**

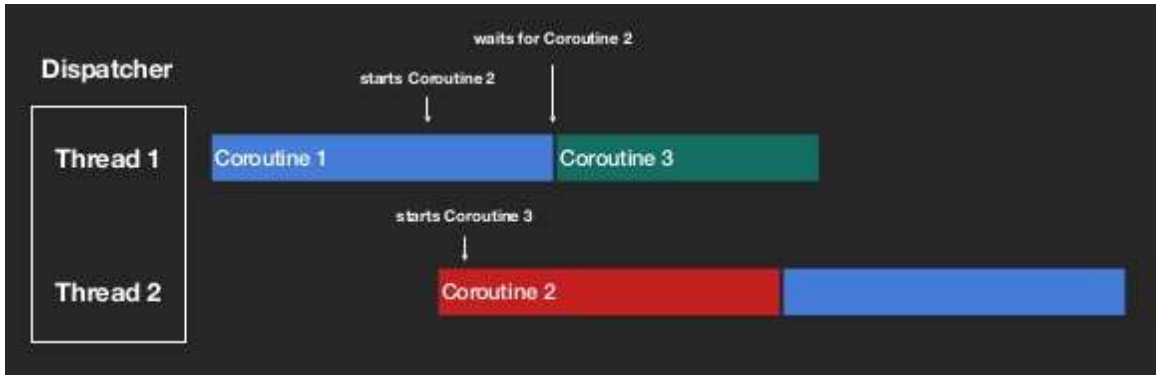


스레드 차단으로 인한 ANR(activity not response) 에러

※ 액티비티가 사용자 이벤트에 대해 5초 이상 무반응시 발생함

❖ Coroutine과 다른 구현 방식의 차이

- 네트워크 통신(request, reply)
  - **AsyncTask(비동기)** : **백그라운드 I/O 스레드**를 통해 수행, 네트워크 요청이후 응답이 오기까지 I/O 스레드는 대기한다. – Deprecated
  - **Rx(리엑트)** : 네트워크 요청 이후 **응답을 관찰(Observable)**하면서 다른 작업을 수행함. 응답이 **관찰**되면 작업을 이어간다. – 멀티 스레드를 다루기 힘들다.(Deadlock)
  - Coroutine : **네트워크 요청**(Coroutine 1) 이후 대기하면서 **다른 Coroutine**(Coroutine 2)을 수행한다. 만약, 네트워크 응답이 올 경우 다시 Coroutine(Coroutine 1)을 수행한다.



< Coroutine 동작 예시 >

www.slideshare.net

Coroutine 활용

❖ Coroutine 스코프(CoroutineScope)

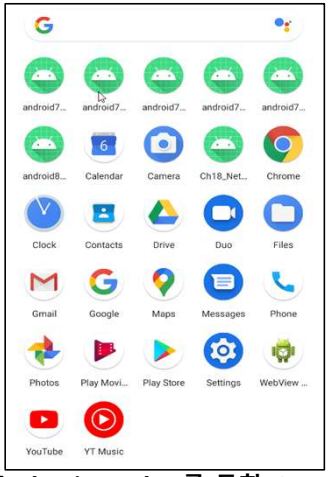
- Coroutine의 **실행 범위(컨텍스트를 인자로 가짐)**
- GlobalScope, ViewModelScope, LifeCycleScope 등이 있다.

❖ Coroutine 컨텍스트(CoroutineContext)

- **동작 목적, 원리가 비슷한 작업을 그룹화**해서 Coroutine을 생성한다.
- **Dispatcher(Thread 할당자)**
  - **Main** : 메인 스레드에서 UI 갱신, Toast, View 등 가벼운 작업을 수행하는 Coroutine
  - **IO** : 네트워킹, 내부 DB 접근 등 백그라운드 스레드에서 필요한 작업을 수행하는 Coroutine
  - **Default** : 크기가 큰 변수를 다루거나 필터링 등 자료구조 및 알고리즘 관련 연산을 수행하는 Coroutine
- **Job**
  - Coroutine에 할당할 Job Instance 이름
  - **Coroutine(Dispatcher.Main + Job1() + Job(2))** : job1, 2를 메인 Coroutine에 할당한다.

❖ Coroutine 빌더(CoroutineBuilder)

- **Launch** : 현재 스레드를 blocking하지 않고 새로운 코루틴을 실행한다.
- **Async** : 수행 이후 결과 값을 **await()**로 반환 받는다. (단, await전까지 해제되지 않음)



Default Dispatcher를 통한 Count와 Main Dispatcher를 통해 뷰를 업데이트 하는 모습

```
val backgroundScope = CoroutineScope( context: Dispatchers.Default + Job())
backgroundScope.launch { this: CoroutineScope
    delay( timeMillis: 1000)
    var time = measureTimeMillis {
        for (i in 1..2_000_000_000) {
            sum += i
            channel.send(sum.toInt())
        }
    }
}
val mainScope = GlobalScope.launch(Dispatchers.Main) { this: CoroutineScope
    channel.consumeEach { it: Int
        binding.resultView.text = "sum : $it"
    }
}
```

CoroutineBuilder

컨텍스트 및 스코프 선언 : 큰 변수를 다루기 위한 Default Dispatcher

Background Task(Count)

컨텍스트 및 스코프 선언 : UI 갱신을 위한 Main Dispatcher

바인딩을 통한 UI 갱신

HTTP 통신 – Volley

❖ Volley 라이브러리

- 구글에서 개발하고 2013년 **공개한 라이브러리로, 쉬운 HTTP 통신 구현**을 가능하게 한다.
- **핵심 요소(클래스)**
  - **RequestQueue** : 서버 요청자
  - **XXXRequest** : XXX 타입(string, image, json, jsonarray) 의 결과를 받는 객체
- **데이터 타입에 따른 요청 방법**
  - **StringRequest** : Method, URL, Listener, ErrorListener 입력
  - **ImageRequest** : URL, Listener, ErrorListener 입력
  - **JsonObjectRequest, JsonArrayRequest** : Method, URL, Listener, ErrorListener 입력
    - Method : 전송 방식(GET, POST)
    - URL : 요청할 대상 주소
    - Listener : 콜백 데이터를 받는 객체
    - ErrorListener : 콜백을 받지 못할 경우 생성되는 객체

요청할 대상 주소

```
val url =
    MyApplication.BASE_URL + "/v2/everything?q=${MyApplication.QUERY}&apiKey=${MyApplication.API_KEY}&pag

val queue = Volley.newRequestQueue(activity)
val jsonRequest =
    object : JsonObjectRequest(
        Request.Method.GET,
        url,
        jsonRequest: null,
        Response.Listener<JSONObject> { response ->
            val jsonArray = response.getJSONArray( name: "articles")
            val mutableList= mutableListOf<ItemModel>()
            for(i in 0 until jsonArray.length()){
                ItemModel().run {  this: ItemModel
                    val article = jsonArray.getJSONObject(i)
                    author=article.getString( name: "author")
                    title=article.getString( name: "title")
                    description=article.getString( name: "description")
                    urlToImage=article.getString( name: "urlToImage")
                    publishedAt=article.getString( name: "publishedAt")
                    mutableList.add(this)  ^run
                }
            }
            binding.volleyRecyclerView.layoutManager=LinearLayoutManager(activity)
            binding.volleyRecyclerView.adapter= MyAdapter(activity as Context, mutableList)
        },
        Response.ErrorListener { error -> println("error.....$error") }){
        override fun getHeaders(): MutableMap<String, String> {
            val map = mutableMapOf<String, String>(
                "User-agent" to MyApplication.USER_AGENT
            )
            return map
        }
    }

queue.add(jsonRequest)
return binding.root
```

전송 방식(GET)

콜백 데이터를 받고 처리하는 객체

정제된 콜백 데이터를 리사이클러뷰로 시각화

에러 발생시 처리하는 객체

생성된 Request를 전송

HTTP 통신 – Retrofit

❖ Retrofit 라이브러리

- 스퀘어에서 개발된 라이브러리로 **구현이 어려워 오랜 시간 숙련을 요구**하지만, **간편한 HTTP 통신**이라는 특징이 있다.
- 코드 가독성이 뛰어나며 **동기/비동기 구현이 가능**하다.
- **Annotation 기반 서비스 인터페이스 동작**
  1. 데이터 전송 방식, 전달 대상 데이터를 인터페이스에 작성한다.
  2. 인터페이스 함수명에 데이터를 입력하여 요청한다.(enqueue)
  3. Response.isSuccessful이면 콜백 데이터를 처리한다.
- Retrofit 구성 요소
  - **DTO** : Data Transfer Object로 콜백 데이터를 JSON 타입으로 변환하여 매핑
  - **Interface** : 사용할 HTTP CRUD 메소드를 정의한다.
  - **Builder** : Interface 사용 인스턴스, 접근 대상 URL, 데이터 변환기 설정

```
val retrofit: Retrofit
    get() = Retrofit.Builder()
        .baseUrl(BASE_URL)
        .addConverterFactory(GsonConverterFactory.create())
        .build()
```

Retrofit Builder Initialization

데이터 전송

```
interface NetworkService {
    @GET( value: "/v2/everything")
    fun getList(
        @Query( value: "q") q: String?,
        @Query( value: "apiKey") apiKey: String?,
        @Query( value: "page") page: Long,
        @Query( value: "pageSize") pageSize: Int
    ): Call<PageListModel>
}
```

반환 받는 데이터 형식

데이터 수신

```
class PageListModel {
    var articles: MutableList<ItemModel>? = null
}
```

CRUD 메소드가 정의되는 Interface

```
val call: Call<PageListModel> = MyApplication.networkService.getList(
    MyApplication.QUERY,
    MyApplication.API_KEY,
    page: 1,
    pageSize: 10
)

call?.enqueue(object : Callback<PageListModel> {
    override fun onResponse(
        call: Call<PageListModel>,
        response: Response<PageListModel>
    ) {
        if (response.isSuccessful()) {
            binding.retrofitRecyclerView.layoutManager= LinearLayoutManager(activity)
            binding.retrofitRecyclerView.adapter= MyAdapter(activity as Context, response.body()?.articles)
        }
    }
```

인터페이스 함수 인자를 입력한 Callable 객체

통신 수행

콜백 데이터 처리

데이터 전송 및 콜백 데이터 처리

기타 편의 기능 – 파일 다운로더

❖ 브로드캐스트 리시버

- 이벤트 모델로 실행되는 컴포넌트
  - 컴포넌트는 인텐트를 통해 실행 가능하기에 **인텐트 필터**와 함께 개발되어야 한다.
- 시스템에 이벤트가 발생시 브로드캐스트 리시버를 통해 해당 **상황에 맞게 앱을 동작** 시킬 수 있다.
- **onReceive()**로만 라이프사이클이 구성되어 있다.
- registerReceiver(), unregisterReceiver()로 리시버의 사용/해제를 설정 가능하다.

❖ 파일 다운로드 기능

- DownloadManager를 통해 외부 파일을 로컬로 다운로드 할 수 있다.
  - DownloadManager는 내부적으로 백그라운드 쓰레드로 동작한다.
- 다운로드가 끝나면 다운로드 매니저가 인텐트를 발생시키고, 브로드캐스트 리시버를 통해 성공 유무를 확인할 수 있다.

다운로드 및 Notification 유형의 인텐트 필터

1.
```
val intentFilter = IntentFilter()
intentFilter.addAction(DownloadManager.ACTION_DOWNLOAD_COMPLETE)
intentFilter.addAction(DownloadManager.ACTION_NOTIFICATION_CLICKED)
registerReceiver(onDownloadComplete, intentFilter)
```

리시버 사용 설정 : 인텐트 발생 시 리시버가 실행되도록 한다.

2.
```
val downloadUrl = "https://cse.pusan.ac.kr/sites/cse/download/cse_newsletter_vol_13_1.pd
val request = DownloadManager.Request(Uri.parse(downloadUrl))
    .setTitle("Downloading a file")
    .setDescription("Downloading CSE Newsletter")
    .setNotificationVisibility(DownloadManager.Request.VISIBILITY_VISIBLE)
    .setDestinationUri(Uri.fromFile(file))
    .setRequiresCharging(false)
    .setAllowedOverMetered(true)
    .setAllowedOverRoaming(true)

downloadId = downloadManager.enqueue(request)
```

DownloadManager Request 설정

설정에 따라 Request 전송

3.
```
private val onDownloadComplete = object : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
```

다운로드 등의 인텐트 발생 시 실행되는 리시버의 onReceive

HTTP 통신 – Glide

❖ Glide 라이브러리

- 구글이 Bump앱을 인수하면서 공개된 라이브러리로, 모든 종류의 이미지를 **빠르게 다운로드** 가능하게 합니다.
- PNG, JPG 뿐만 아니라 **GIF 이미지의 애니메이션 효과**를 함께 출력한다.
- 이미지 뷰의 크기에 맞게 **자동으로 이미지 용량을 줄여**서 출력한다.
- **Placeholder, error 기능을 함께 제공**하여 개발자의 실수를 방지한다.
  - **Placeholder** : load된 이미지가 아직 없을 때 대체 이미지
  - **Error** : load 대상 이미지가 경로에 없을 때 대체 이미지

```
Glide.with(context)
    .load(R.drawable.img1)
    .into(binding.itemImage)
```

리소스 이미지 출력

```
Glide.with(context)
    .load(model.urlToImage)
    .into(binding.itemImage)
```

서버 이미지 출력

```
Glide.with(context)
    .load(model.urlToImage)
    .placeholder(R.drawable.loading)
    .error(R.drawable.error)
    .into(binding.itemImage)
```

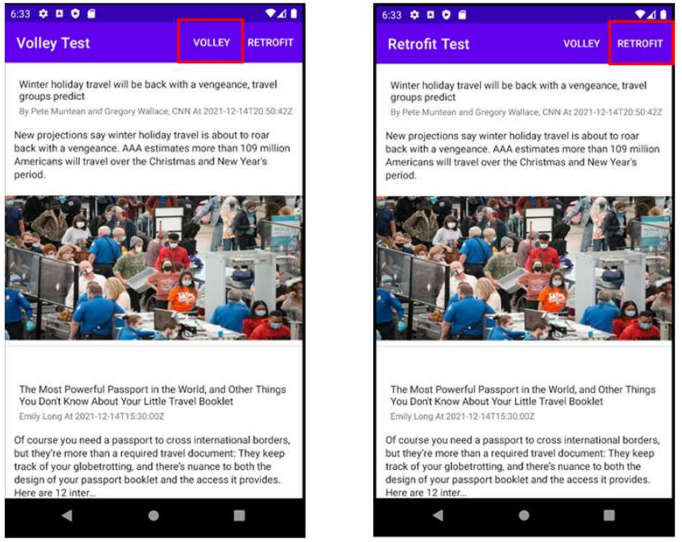
Placeholder, Error 대체 기능

실습

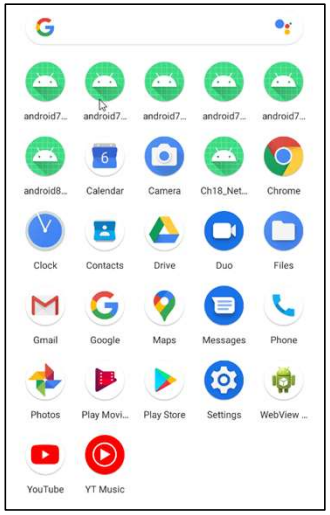
실습

- 실습
  - 예제 1 – 정컴 소식지 다운로드
  - 예제 2 – Volley, Retrofit, Glide를 활용한 뉴스 앱
  - 예제 3 – Coroutine 기반 스톱워치

정컴소식지 다운로드 파일



뉴스 앱 구현 결과(좌 : Volley, 우 : Retrofit)



Coroutine 구현 결과 : 실시간 뷰 변경

예제 1 – 정컴 소식지 다운로드

❖ 정컴 소식지 다운로드 해보기

1. 메인 레이아웃 배치
2. 브로드캐스트 리시버 구현
3. 인텐트 필터 및 리시버 연결부 구현

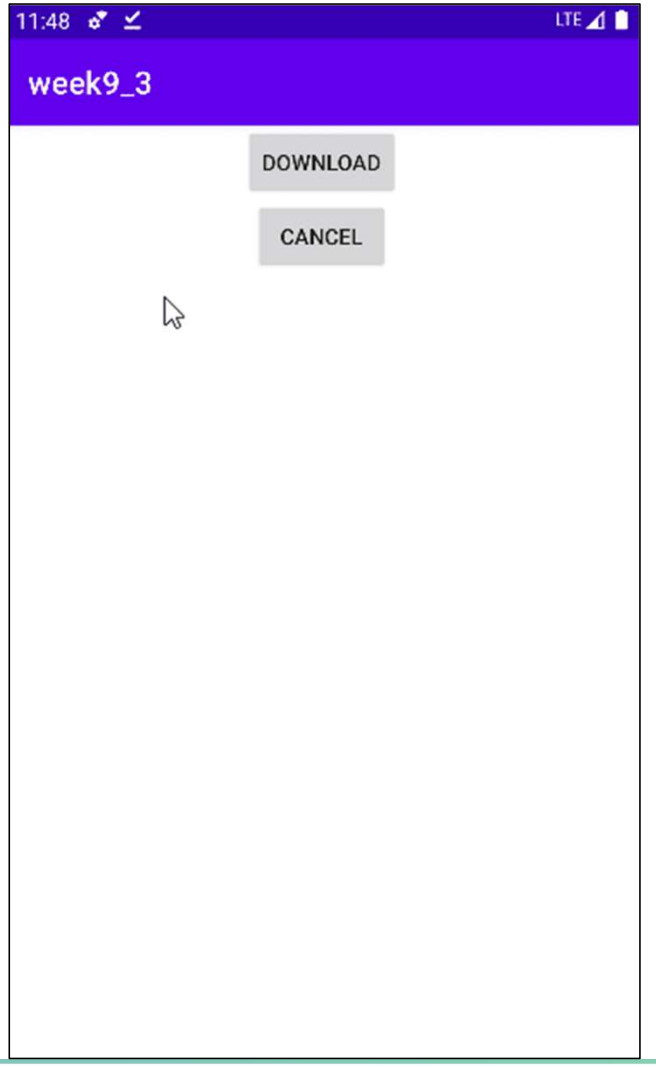
Build.gradle (:app)

```
6   android {
7       buildFeatures {
8           viewBinding = true
9       }
```

추가하기

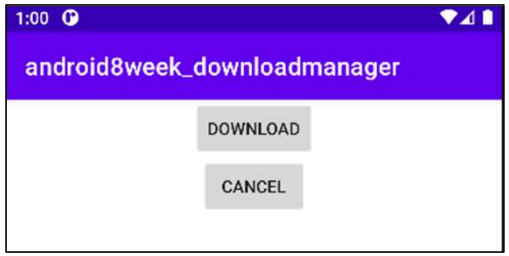
AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
```



예제 1 – 정컴 소식지 다운로드

1. 메인 레이아웃 배치



activity_main.xml

```xml
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/downloadBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Download"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>
    <Button
        android:id="@+id/cancelBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Cancel"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/downloadBtn"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

예제 1 – 정컴 소식지 다운로드

2. 브로드캐스트 리시버 구현

- (23 line) : 브로드캐스트 리시버 선언
- (24-46 line) : 리시버 onReceive 구현

MainActivity.kt

```
19  class MainActivity : AppCompatActivity() {
20      private var downloadId: Long = -1L
21      private lateinit var downloadManager: DownloadManager
22
23      private val onDownloadComplete = object : BroadcastReceiver() {
24          override fun onReceive(context: Context, intent: Intent) {
25              val id = intent.getLongExtra(DownloadManager.EXTRA_DOWNLOAD_ID,  defaultValue: -1)
26              if (DownloadManager.ACTION_DOWNLOAD_COMPLETE.equals(intent.action)) {
27                  if (downloadId == id) {
28                      val query: DownloadManager.Query = DownloadManager.Query()
29                      query.setFilterById(id)
30                      var cursor = downloadManager.query(query)
31                      if (!cursor.moveToFirst()) {
32                          return
33                      }
34
35                      var columnIndex = cursor.getColumnIndex(DownloadManager.COLUMN_STATUS)
36                      var status = cursor.getInt(columnIndex)
37                      if (status == DownloadManager.STATUS_SUCCESSFUL) {
38                          Toast.makeText(context,  text: "Download succeeded", Toast.LENGTH_SHORT).show()
39                      } else if (status == DownloadManager.STATUS_FAILED) {
40                          Toast.makeText(context,  text: "Download failed", Toast.LENGTH_SHORT).show()
41                      }
42                  }
43              } else if (DownloadManager.ACTION_NOTIFICATION_CLICKED.equals(intent.action)) {
44                  Toast.makeText(context,  text: "Notification clicked", Toast.LENGTH_SHORT).show()
45              }
46          }
47      }
```

예제 1 – 정컴 소식지 다운로드

3. 인텐트 필터 및 리시버 연결부 구현

- (54 line) : 다운로드 매니저 객체
- (56-58 line) : 인텐트 필터 선언
- (59 line) 리시버와 인텐트 필터 연결
- (65-72 line) 다운로드 매니저 Request 설정
- (74 line) 설정된 Request 전송
- (84-89 line) 리시버와 인텐트 필터 연결 해제

MainActivity.kt

```
84    override fun onDestroy() {
85        super.onDestroy()
86        unregisterReceiver(onDownloadComplete)
87    }
88
89    }
```

MainActivity.kt

```
49    override fun onCreate(savedInstanceState: Bundle?) {
50        var binding = ActivityMainBinding.inflate(layoutInflater)
51        super.onCreate(savedInstanceState)
52        setContentView(binding.root)
53
54        downloadManager = getSystemService(Context.DOWNLOAD_SERVICE) as DownloadManager
55
56        Todo
57
58
59
60
61        binding.downloadBtn.setOnClickListener { it: View!
62            val file = File(Environment.getExternalStoragePublicDirectory(
63                type: Environment.DIRECTORY_DOWNLOADS + "/"), child: "newsletter.pdf")
64            val downloadUrl = "https://cse.pusan.ac.kr/sites/cse/download/201912_cse_newsletter_vol_29.pdf"
65            Todo
66
67
68
69
70
71
72
73
74
75        }
76
77        binding.cancelBtn.setOnClickListener { it: View!
78            Todo
79
80        }
81    }
82
```

원하는 주소 입력

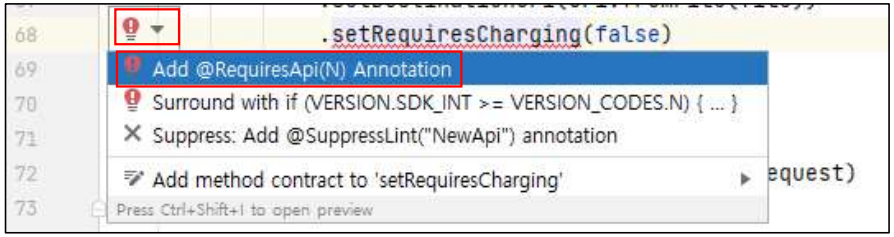
예제 1 – 정컴 소식지 다운로드

3. 인텐트 필터 및 리시버 연결부 구현

- API Level 변경 방법
  - **방법 1)** Add Anotation 추가
  - **방법 2)** minSdk 값 변경

방법 1)

MainActivity.kt



MainActivity.kt

```
@RequiresApi(Build.VERSION_CODES.N)
override fun onCreate(savedInstanceState: Bundle?) {
    var binding = ActivityMainBinding.inflate(layoutInflater)
    super.onCreate(savedInstanceState)
    setContentView(binding.root)
```

방법 2)

Build.gradle (:app)

```
android {
    buildFeatures{
        viewBinding = true
    }
    compileSdk 32

    defaultConfig {
        applicationId "com.example.week9_3"
        minSdk 30
```

수정하기

예제 2 – 뉴스 앱 만들기(기반 작업, Glide 활용)

❖ 뉴스 앱 만들기(기반 작업, Glide 활용)

1. 실습자료 다운로드
2. 뉴스 리스트가 출력될 리사이클러뷰의 어댑터 구현
3. Retrofit 실행을 위한 설정
4. Retrofit 쿼리 구현
5. 콜백 결과 데이터 정의
6. Retrofit으로 출력될 뉴스 페이지
7. Volley로 출력될 뉴스 페이지

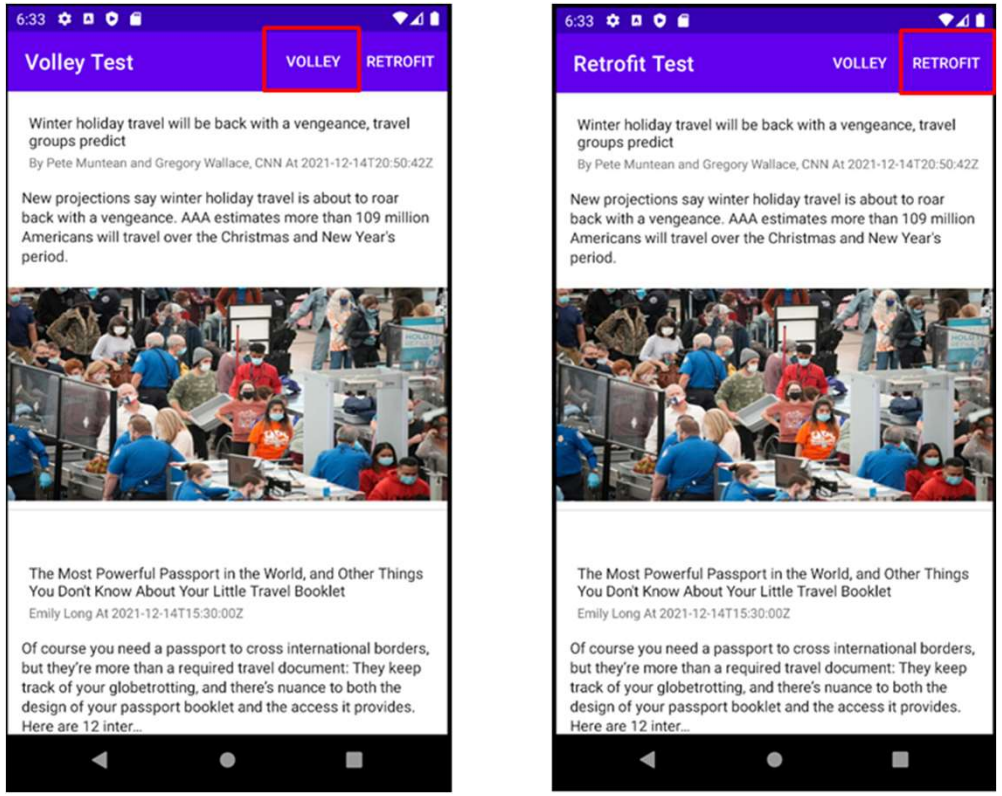
```
android {
    buildFeatures{
        viewBinding = true
        dataBinding = true
    }
```

```
implementation 'com.android.volley:volley:1.1.1'
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.google.code.gson:gson:2.8.6'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
implementation 'com.github.bumptech.glide:glide:4.11.0'
```

build.gradle 코드 추가 사항

```
<uses-permission android:name="android.permission.INTERNET" />
```

Manifest 코드 추가 사항



뉴스 앱 구현 결과(좌 : Volley, 우 : Retrofit)

❖ 뉴스 앱 만들기(기반 작업, Glide 활용)

- 뉴스 리스트가 출력될 리사이클러뷰의 어댑터 구현
  - (27-29 line) : Glide 활용

recylcer/MyAdapter.kt

```
12  class MyViewHolder(val binding: ItemMainBinding): RecyclerView.ViewHolder(binding.root)
13  class MyAdapter(val context: Context, val datas: MutableList<ItemModel>?):
14      RecyclerView.Adapter<RecyclerView.ViewHolder>(){
15
16      override fun getItemCount(): Int{return datas?.size ?: 0}
17
18      override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder
19              = MyViewHolder(ItemMainBinding.inflate(LayoutInflater.from(parent.context), parent,  attachToParent: false))
20
21      override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
22          val binding=(holder as MyViewHolder).binding
23          val model=datas!![position]
24          binding.itemTitle.text=model.title
25          binding.itemDesc.text=model.description
26          binding.itemTime.text="${model.author} At ${model.publishedAt}"
27          Glide.with(context)
28
29              Todo
30      }
31  }
```

예제 2 – 뉴스 앱 만들기(기반 작업, Glide 활용)

❖ Retrofit 실행을 위한 설정

- (11-14 line) : 쿼리 전송을 위한 요구 데이터
- (12 line) : API_KEY 는 https://newsapi.org/ 에서 개별로 획득 후 사용 (다음 페이지 참조)
- (17-21 line) : Retrofit Builder 선언
- (23 line) : Retrofit Initialization

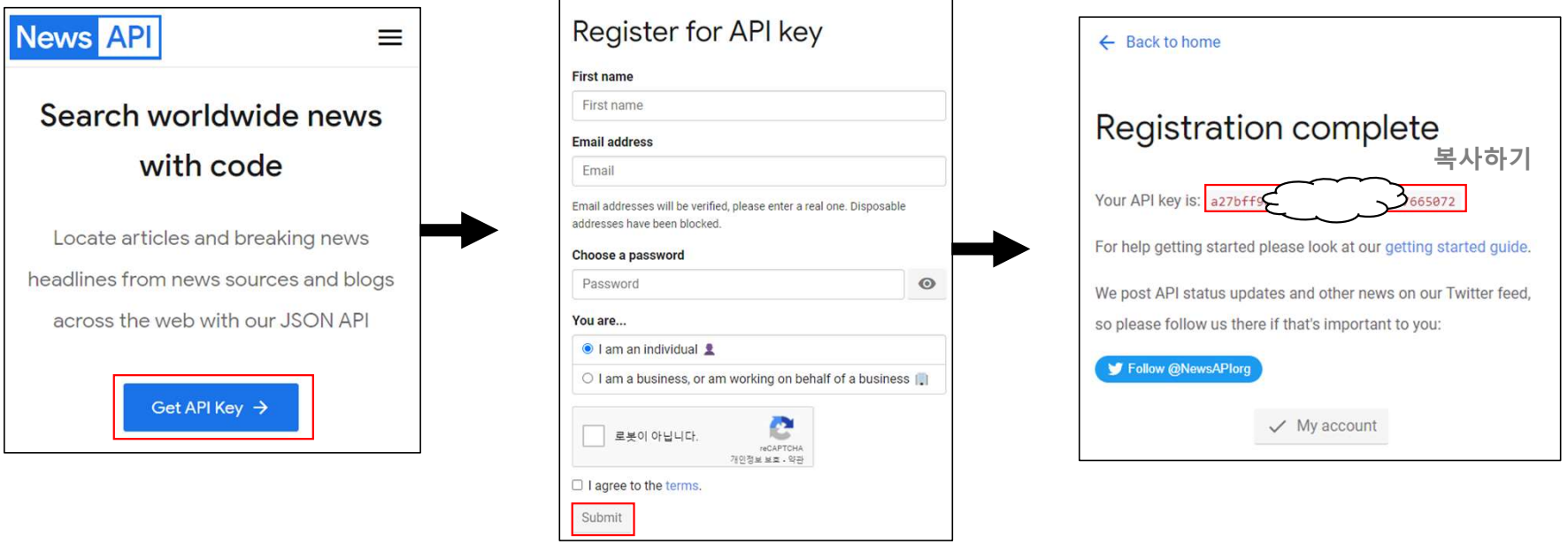
MyApplication.kt

```
9   class MyApplication: Application() {
10      companion object {
11          val QUERY = "travel"
12          val API_KEY = "079[redacted]854b7"
13          val BASE_URL = "https://newsapi.org"
14          val USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36"
15
16          var networkService: NetworkService
17          val retrofit: Retrofit
18              get() = Retrofit.Builder()
19                  .baseUrl(BASE_URL)
20                  .addConverterFactory(GsonConverterFactory.create())
21                  .build()
22          init {
23              networkService = retrofit.create(NetworkService::class.java)
24          }
25      }
26
27  }
```

예제 2 – 뉴스 앱 만들기(기반 작업, Glide 활용)

❖ Retrofit 실행을 위한 설정

- https://newsapi.org/ 접속
- API Key 등록
- API key 복사



예제 2 – 뉴스 앱 만들기(기반 작업, Glide 활용)

❖ Retrofit 쿼리 구현

- (9 line) : GET Method를 통한 데이터 통신
- (10-15 line) : 데이터를 요청하는 쿼리문, 콜백 결과로 PageListModel 데이터 반환

```
8   interface NetworkService {
9       @GET( value: "/v2/everything")
10      fun getList(
11          @Query( value: "q") q: String?,
12          @Query( value: "apiKey") apiKey: String?,
13          @Query( value: "page") page: Long,
14          @Query( value: "pageSize") pageSize: Int
15      ): Call<PageListModel>
16  }
```

retrofit/NetworkService 코드

❖ 콜백 결과 데이터 정의

```
3   class PageListModel {
4       var articles: MutableList<ItemModel>? = null
5   }
```

model/PageListModel 코드

```
3   class ItemModel {
4       var id: Long = 0
5       var author: String? = null
6       var title: String? = null
7       var description: String? = null
8       var urlToImage: String? = null
9       var publishedAt: String? = null
10  }
```

model/ItemModel 코드

예제 2 – 뉴스 앱 만들기(기반 작업, Glide 활용)

RetrofitFragment.kt

❖ **Retrofit으로 출력될 뉴스 페이지**

- (25-29 line) : 인터페이스 기반 Callable 객체
- (31 line) : enqueue로 통신 수행
- (36-39 line) : 콜백 데이터 처리
  - 7주차 참고
- (43-45 line) : 콜백 실패 시 데이터 처리

```
18  class RetrofitFragment : Fragment() {
19      override fun onCreateView(
20          inflater: LayoutInflater, container: ViewGroup?,
21          savedInstanceState: Bundle?
22      ): View? {
23          val binding = FragmentRetrofitBinding.inflate(inflater, container, attachToParent: false)
24          //.............................
25          val call: Call<PageListModel> = MyApplication.networkService.getList(
26              MyApplication.QUERY,
27              MyApplication.API_KEY,
28              page: 1,
29              pageSize: 10
30          )
31          call?.enqueue(object : Callback<PageListModel> {
32              override fun onResponse(
33                  call: Call<PageListModel>,
34                  response: Response<PageListModel>
35              ) {
36                  if (response.isSuccessful()) {
37
38                      Recycler View 관련 처리
39
40                  }
41              }
42
43              override fun onFailure(
44                  call: Call<PageListModel?>,
45                  t: Throwable
46              ) {
47              }
48          })
49          return binding.root
50      }
51  }
```

예제 2 – 뉴스 앱 만들기(기반 작업, Glide 활용)

❖ **Volley로 출력될 뉴스 페이지**

- (56-57 line) : 처리된 데이터를 RecyclerView로 시각화
  - 7주차 참고
- (59-64 line) : Request 에러 처리
- (68 line) : Request 전송

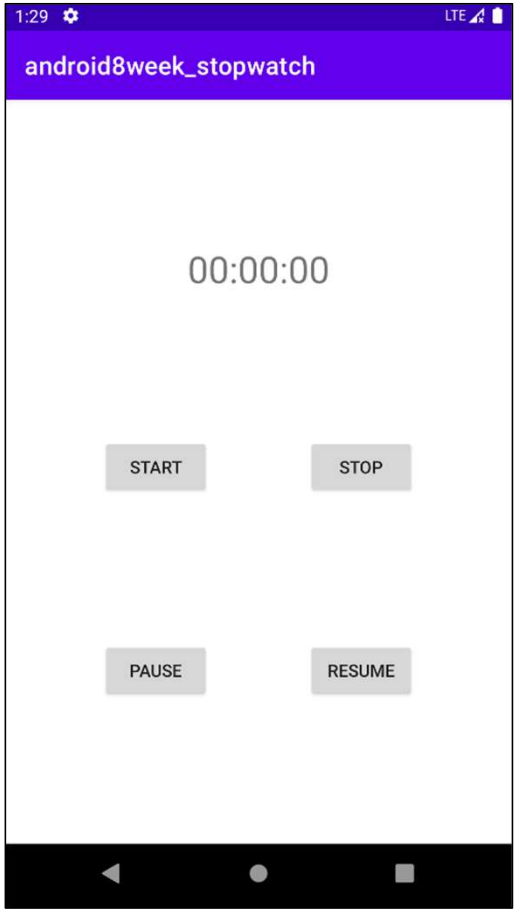
VollyFragment.kt

```
53                      }
54                  }
55
56                      Recycler View 관련 처리
57
58              },
59              Response.ErrorListener { error -> println("error.....$error") }){
60              override fun getHeaders(): MutableMap<String, String> {
61                  val map = mutableMapOf<String, String>(
62                      "User-agent" to MyApplication.USER_AGENT
63                  )
64                  return map
65              }
66          }
67
68          queue.add(jsonRequest)
69          return binding.root
70
71      }
72
73  }
```

예제 3 – Coroutine 기반 스톱워치

❖ Coroutine 기반 스톱워치

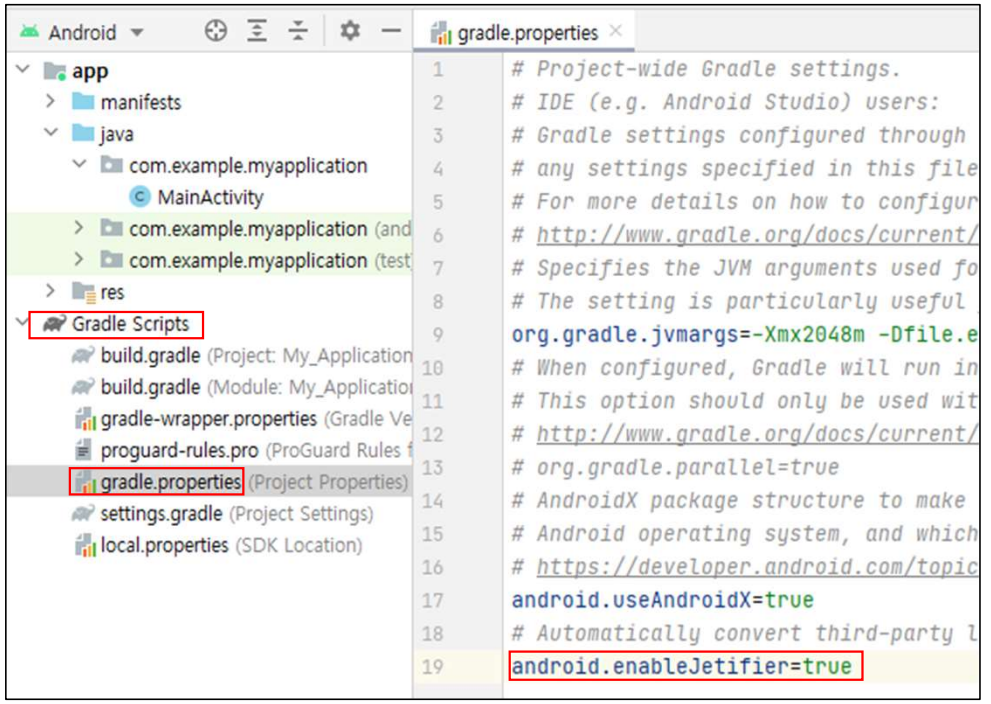
- 시작, 정지, 일시정지, 계속 기능이 있는 스톱워치 만들기
- 조건
  - 우측 화면과 동일(stop 상태에서 start가 동작, pause 상태에서 resume이 동작하도록 설계)
  - Coroutine은 sleep 대신 delay를 사용
  - Default Dispatcher에 시간 측정을 할당하고, Main Dispatcher를 통해 결과가 뷰에 반영 되는 계산기



에러 – Duplication class

❖ Duplication class

- 앱 빌드 시 내부 라이브러리 충돌로 발생하는 에러
- 해결방법 1 – Build.gradle 에서 라이브러리 중복 제거하기
- 해결방법 2 – 바이너리 재 작성 및 자동 연결 설정하기
  - Gradle Scripts – gradle.properties 에서 “android.enableJetifier=true” 추가



에러 – permission

- AndroidManifest.xml
  - <uses-permission android:name="android.permission.INTERNET"/> 추가할 것