세상의 속도를
따라잡고 싶다면

Do it!

깡샘의 안드로이드 앱 프로그래밍 with 코틀린

이지스퍼블리싱(주)

18

네트워크 프로그래밍

이지스퍼블리싱(주)

18-1 스마트폰 정보 구하기

전화 상태 변화 감지하기 – PhoneStateListener

- 스마트폰의 상태를 파악하는 방법은 PhoneStateListener를 이용하는 방법과 TelephonyCallback을 이용하는 방법이 있습니다.
- PhoneStateListener를 이용하는 방법은 안드로이드 초기 버전부터 제공되던 방식인데 안드로이드 12(API 레벨 31)에서 deprecated 되었고, 대신 TelephonyCallback이 추가
- PhoneStateListener를 이용하려면 PhoneStateListener를 상속받은 클래스를 작성하고 그 클래스의 객체를 TelephonyManager에 등록
  - onCallForwardingIndicatorChanged(boolean cfi): 통화 전달 상태 변경
  - onCallStateChanged(int state, String incomingNumber): 통화 상태 변경
  - onCellLocationChanged(CellLocation location): 폰의 기지국 위치 변경
  - onDataActivity(int direction): 데이터 송수신 활동
  - onDataConnectionStateChanged(int state, int networkType): 데이터 연결 상태 변경
  - onMessageWaitingIndicatorChanged(boolean mwi): 메시지 대기 상태 변경
  - onServiceStateChanged(ServiceState serviceState): 단말기의 서비스 상태 변경
  - onSignalStrengthsChanged(SignalStrength signalStrength): 신호 세기 변경

18-1 스마트폰 정보 구하기

- TelephonyManager 객체를 얻고 이 객체의 listen() 함수에 PhoneStateListener 객체를 등록

• 상태 변화 감지

```
val phoneStateListener = object : PhoneStateListener() {
    override fun onServiceStateChanged(serviceState: ServiceState?) {
        super.onServiceStateChanged(serviceState)
        (... 생략 ...)
    }
}
```

• 전화 매니저 얻기

```
val manager = getSystemService(Context.TELEPHONY_SERVICE) as TelephonyManager
manager.listen(phoneStateListener, PhoneStateListener.LISTEN_SERVICE_STATE)
```

18-1 스마트폰 정보 구하기

- PhoneState Listener 상수
  - LISTEN_CALL_FORWARDING_INDICATOR: 통화 전달 지시자
  - LISTEN_CALL_STATE: 통화 상태
  - LISTEN_CELL_LOCATION: 기지국 위치
  - LISTEN_DATA_ACTIVITY: 데이터 송수신 활동
  - LISTEN_DATA_CONNECTION_STATE: 데이터 연결 상태
  - LISTEN_MESSAGE_WAITING_INDICATOR: 메시지 대기 지시자
  - LISTEN_SERVICE_STATE: 단말기의 서비스 상태
  - LISTEN_SIGNAL_STRENGTHS: 신호 세기

- 여러 개의 상태를 함께 감지

• 여러 상태 감지

```
manager.listen(phoneStateListener, PhoneStateListener.LISTEN_SERVICE_STATE or
    PhoneStateListener.LISTEN_CALL_STATE)
```

18-1 스마트폰 정보 구하기

- 상태 변화 감지를 해제할 때는 listen() 함수에 LISTEN_NONE 상수를 지정

• 상태 감지 해제

```
manager.listen(phoneStateListener, PhoneStateListener.LISTEN_NONE)
```

- onServiceStateChanged() 함수는 비행 모드나 긴급 통화 등 스마트폰의 서비스 상태가 바뀌는 순간에 호출
  - STATE_IN_SERVICE: 서비스 가능 상태
  - STATE_EMERGENCY_ONLY: 긴급 통화만 가능한 상태
  - STATE_OUT_OF_SERVICE: 서비스 불가 상태
  - STATE_POWER_OFF: 비행 모드 등 전화 기능을 꺼 놓은 상태

• 서비스 상태 감지

```
override fun onServiceStateChanged(serviceState: ServiceState?) {
    when (serviceState?.state) {
        ServiceState.STATE_EMERGENCY_ONLY -> Log.d("kkang", "EMERGENCY_ONLY....")
        ServiceState.STATE_OUT_OF_SERVICE -> Log.d("kkang", "OUT_OF_SERVICE....")
        ServiceState.STATE_POWER_OFF -> Log.d("kkang", "POWER_OFF....")
        ServiceState.STATE_IN_SERVICE -> Log.d("kkang", "IN_SERVICE....")
    }
}
```

18-1 스마트폰 정보 구하기

- 전화가 걸려오는 상태를 감지하는 onCallStateChanged() 함수
  - CALL_STATE_IDLE: 통화 대기 상태
  - CALL_STATE_RINGING: 벨이 울리는 상태
  - CALL_STATE_OFFHOOK: 통화 중인 상태

• 전화가 걸려오는 상태 감지

```
override fun onCallStateChanged(state: Int, phoneNumber: String?) {
    when (state) {
        TelephonyManager.CALL_STATE_IDLE -> Log.d("kkang", "IDLE..")
        TelephonyManager.CALL_STATE_RINGING -> Log.d("kkang", "RINGING..")
        TelephonyManager.CALL_STATE_OFFHOOK -> Log.d("kkang", "OFFHOOK..")
    }
}
```

18-1 스마트폰 정보 구하기

전화 상태 변화 감지하기 – TelephonyCallback

- 안드로이드 12 버전부터는 스마트폰의 각종 상태 변화를 감지할 때 TelephonyCallback이용

• 스마트폰 상태 퍼미션

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

• TelephonyCallback을 이용한 전화 상태 변화 감지

```
telephonyManager = getSystemService(Context.TELEPHONY_SERVICE) as TelephonyManager
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
    telephonyManager.registerTelephonyCallback(
        mainExecutor,
        object : TelephonyCallback(), TelephonyCallback.CallStateListener {
            override fun onCallStateChanged(state: Int) {
                when (state) {
                    TelephonyManager.CALL_STATE_IDLE -> {
                        Log.d("kkang", "IDLE")
                    }
                    TelephonyManager.CALL_STATE_OFFHOOK -> {
                        Log.d("kkang", "OFFHOOK")
                    }
                    TelephonyManager.CALL_STATE_RINGING -> {
                        Log.d("kkang", "RINGING")
                    }
                }
            }
        })
}
```

18-1 스마트폰 정보 구하기

- 각종 상태 변화를 감지하는 인터페이스
  - TelephonyCallback.CellLocationListener: 셀(cell) 위치 변화 감지
  - TelephonyCallback.ServiceStateListener: 서비스 상태 변화 감지
  - TelephonyCallback.SignalStrengthsListener: 신호 세기 변화 감지
  - TelephonyCallback.DataActivityListener: 데이터 송수신 상태 변화 감지
  - TelephonyCallback.DataConnectionStateListener: 데이터 접속 상태 변화 감지
  - TelephonyCallback.CallStateListener: 전화 상태 변화 감지

18-1 스마트폰 정보 구하기

네트워크 제공 국가, 사업자, 전화번호 얻기 - TelephonyManager

- TelephonyManager는 네트워크 제공 국가, 사업자, 전화번호 등을 반환하는 다음 함수도 제공
  - getNetworkCountryIso(): 네트워크 제공 국가
  - getNetworkOperatorName(): 네트워크 제공 사업자
  - getLine1Number(): 스마트폰의 전화번호

• 사용자 전화번호를 읽는 퍼미션

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

• 네트워크 국가, 사업자, 전화번호 얻기

```
val countryIso = telephonyManager.networkCountryIso
val operatorName = telephonyManager.networkOperatorName
val phoneNumber = telephonyManager.line1Number
```

18-1 스마트폰 정보 구하기

네트워크 접속 정보 - ConnectivityManager

• 네트워크 상태 접근 퍼미션

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

- getActiveNetwork() 함수를 이용하는 방법
  - API 레벨 23부터 제공
  - 23 하위 버전에서도 실행되는 앱을 개발한다면 ConnectivityManager의 getActiveNetworkInfo() 함수를 이용해 NetworkInfo 객체를 얻어야 합니다.

18-1 스마트폰 정보 구하기

• 네트워크 접속 가능 여부

```
private fun isNetworkAvailable(): Boolean {
    val connectivityManager = getSystemService(Context.CONNECTIVITY_SERVICE)
        as ConnectivityManager
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        val nw = connectivityManager.activeNetwork ?: return false
        val actNw = connectivityManager.getNetworkCapabilities(nw) ?: return false
        return when {
            actNw.hasTransport(NetworkCapabilities.TRANSPORT_WIFI) -> {
                Log.d("kkang", "wifi available")
                true
            }
            actNw.hasTransport(NetworkCapabilities.TRANSPORT_CELLULAR) -> {
                Log.d("kkang", "cellular available")
                true
            }
            else -> false
        }
    } else {
        return connectivityManager.activeNetworkInfo?.isConnected ?: false
    }
}
```

18-1 스마트폰 정보 구하기

- requestNetwork() 함수를 이용하는 방법

• CHANGE_NETWORK_STATE 퍼미션

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

- NetworkRequest의 addCapability()와 addTransportType() 함수를 이용하면 네트워크 타입을 지정

• 네트워크 타입 지정

```
val networkReq: NetworkRequest = NetworkRequest.Builder()
    .addCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET)
    .addTransportType(NetworkCapabilities.TRANSPORT_CELLULAR)
    .addTransportType(NetworkCapabilities.TRANSPORT_WIFI)
    .build()
```

18-1 스마트폰 정보 구하기

- NetworkRequest 객체에 네트워크 타입을 설정한 후 객체를 requestNetwork() 함수의 두 번째 매개변수로 지정

• 네트워크 접속 가능 여부

```
conManager.requestNetwork(networkReq, object : ConnectivityManager.NetworkCallback() {
    override fun onAvailable(network: Network) {
        super.onAvailable(network)
        Log.d("kkang", "NetworkCallback...onAvailable....")
    }
    override fun onUnavailable() {
        super.onUnavailable()
        Log.d("kkang", "NetworkCallback...onUnavailable....")
    }
})
```

18-2 HTTP 통신하기

• 인터넷 퍼미션 선언

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

- 안드로이드 앱은 네트워크 통신을 할 때 기본으로 HTTPS 보안 프로토콜을 사용
- 만약 일반 HTTPS 프로토콜로 통신하려면 특정 도메인만 허용하도록 선언

• HTTP 통신 허용

```
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">xxx.xxx.xxx.xxx</domain>
    </domain-config>
</network-security-config>
```

HTTP 프로토콜로 접속을 허용할 IP나 도메인

18-2 HTTP 통신하기

- XML 파일을 매니페스트의 <application> 태그에 networkSecurityConfig 속성으로 지정

• 매니페스트에 HTTP 허용 XML 등록

```
<application
    (... 생략 ...)
    android:networkSecurityConfig="@xml/network_security_config">
```

- 또는 매니페스트에서 usesCleartextTraffic 속성을 true로 설정하면 앱 전체에서 모든 도메인의 서버와 HTTP 통신을 할 수 있습니다.

• 모든 HTTP 통신 허용

```
<application
    (... 생략 ...)
    android:usesCleartextTraffic="true">
```

18-2 HTTP 통신하기

Volley 라이브러리

- 구글에서 제공하는 Volley와 스퀘어에서 제공하는 Retrofit
- Volley는 2013년 구글 IO 행사에서 공개된 라이브러리

• Volley 라이브러리 등록

```
implementation 'com.android.volley:volley:1.2.1'
```

- Volley에서 핵심 클래스
  - RequestQueue: 서버 요청자
  - XXXRequest: XXX 타입의 결과를 받는 요청 정보

18-2 HTTP 통신하기

- 문자열 데이터 요청하기 — StringRequest
  - StringRequest(int method, String url, Response.Listener<String> listener, Response.ErrorListener errorListener)

• 문자열 요청 정의

```
val stringRequest = StringRequest(
    Request.Method.GET,
    url,
    Response.Listener<String> {
        Log.d("kkang", "server data : $it")
    },
    Response.ErrorListener { error ->
        Log.d("kkang", "error.....$error")
    })
```

• 서버에 요청하기

```
val queue = Volley.newRequestQueue(this)
queue.add(stringRequest)
```

18-2 HTTP 통신하기

- POST 방식에서는 StringRequest를 상속받은 클래스를 이용

• POST 방식으로 데이터 전송

```
val stringRequest = object : StringRequest(
    Request.Method.POST,
    url,
    Response.Listener<String> {
        Log.d("kkang", "server data : $it")
    },
    Response.ErrorListener { error ->
        Log.d("kkang", "error.....$error")
    }){
    override fun getParams(): MutableMap<String, String> {
        return mutableMapOf<String, String>("one" to "hello", "two" to "world")
    }
}
```

18-2 HTTP 통신하기

- 이미지 데이터 요청하기 — ImageRequest
  - public ImageRequest(String url, Response.Listener<Bitmap> listener, int maxWidth, int maxHeight, ScaleType scaleType, Config decodeConfig, @Nullable Response.ErrorListener errorListener)

  - url: 서버 URL
  - listener: 결과를 가져오는 콜백
  - maxWidth, maxHeight: 지정한 값으로 이미지 크기 조절해서 전달. 만약 0으로 설정하면 크기 조절 없이 서버가 전달하는 이미지를 그대로 받음
  - scaleType: 영역에 맞게 이미지의 크기를 확대 또는 축소하는 스케일 타입
  - decodeConfig: 이미지 형식 지정
  - errorListener: 오류 콜백

18-2 HTTP 통신하기

• 이미지 요청 정의

```
val imageRequest = ImageRequest(
    url,
    Response.Listener { response -> binding.imageView.setImageBitmap(response) },
    0,
    0,
    ImageView.ScaleType.CENTER_CROP,
    null,
    Response.ErrorListener { error ->
        Log.d("kkang", "error.............$error")
    })

val queue = Volley.newRequestQueue(this)
queue.add(imageRequest)
```

18-2 HTTP 통신하기

- 화면 출력용 이미지 데이터 요청하기 — NetworkImageView
  - NetworkImageView는 Volley에서 제공하는 이미지 출력용 뷰
  - setImageUrl(String url, ImageLoader imageLoader)

• 화면 출력용 이미지 데이터 요청하기

```
<com.android.volley.toolbox.NetworkImageView
    android:id="@+id/networkImageView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

• setImageUrl() 함수로 요청하기

```
val queue = Volley.newRequestQueue(this)
val imgMap = HashMap<String, Bitmap>()
imageLoader = ImageLoader(queue, object : ImageLoader.ImageCache {
    override fun getBitmap(url: String): Bitmap? {
        return imgMap[url]
    }
    override fun putBitmap(url: String, bitmap: Bitmap) {
        imgMap[url] = bitmap
    }
})
binding.networkImageView.setImageUrl(url, imageLoader)
```

18-2 HTTP 통신하기

- JSON 데이터 요청하기 — JsonObjectRequest
  - JsonObjectRequest(int method, String url, JSONObject jsonRequest, Response.Listener<JSONObject> listener, Response.ErrorListener errorListener)

• JSON 데이터 요청하기

```
val jsonRequest =
    JsonObjectRequest(
        Request.Method.GET,
        url,
        null,
        Response.Listener<JSONObject> {  response ->
            val title = response.getString("title")
            val date = response.getString("date")
            Log.d("kkang","$title, $date")
        },
        Response.ErrorListener { error -> Log.d("kkang","error....$error")
    })
val queue = Volley.newRequestQueue(this)
queue.add(jsonRequest)
```

18-2 HTTP 통신하기

- JSON 배열 요청하기 — JsonArrayRequest
  - JsonArrayRequest(String url, JSONArray jsonRequest, Response.Listener<JSONArray> listener, Response.ErrorListener errorListener)

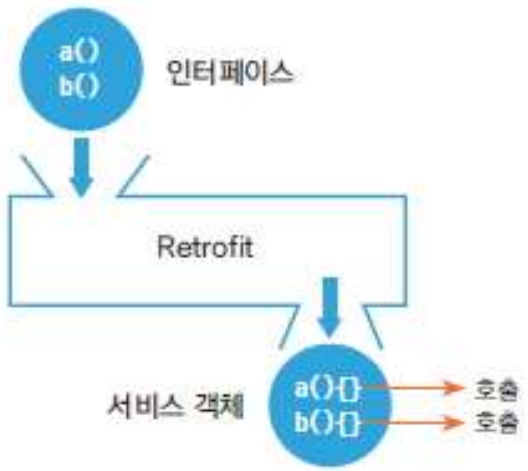
• JSON 배열 요청하기

```
val jsonArrayRequest = JsonArrayRequest(
    Request.Method.GET,
    url,
    null,
    Response.Listener<JSONArray> { response ->
        for (i in 0 until response.length()) {
            val jsonObject = response[i] as JSONObject
            val title = jsonObject.getString("title")
            val date = jsonObject.getString("date")
            Log.d("kkang","$title, $date")
        }
    },
    Response.ErrorListener { error -> Log.d("kkang", "error....$error") }
)
val queue = Volley.newRequestQueue(this)
queue.add(jsonArrayRequest)
```

18-2 HTTP 통신하기

Retrofit 라이브러리

- Retrofit*은 스퀘어에서 만든 HTTP 통신을 간편하게 만들어 주는 라이브러리
- Retrofit은 네트워크 통신 정보만 주면 그대로 네트워크 프로그래밍을 대신 구현
- Retrofit 동작 방식
  1. 통신용 함수를 선언한 인터페이스를 작성합니다.
  2. Retrofit에 인터페이스를 전달합니다.
  3. Retrofit이 통신용 서비스 객체를 반환합니다.
  4. 서비스의 통신용 함수를 호출한 후 Call 객체를 반환합니다.
  5. Call 객체의 enqueue() 함수를 호출하여 네티워크 통신을 수행합니다.



18-2 HTTP 통신하기

- 라이브러리 선언
  - Retrofit은 JSON이나 XML 데이터를 모델(VO 클래스) 객체로 변환
  - JSON, XML을 파싱하는 라이브러리가 필요
  - 파싱 라이브러리에 맞는 converter 라이브러리 필요
    - Gson: com.squareup.retrofit2:converter-gson
    - Jackson: com.squareup.retrofit2:converter-jackson
    - Moshi: com.squareup.retrofit2:converter-moshi
    - Protobuf: com.squareup.retrofit2:converter-protobuf
    - Wire: com.squareup.retrofit2:converter-wire
    - Simple XML: com.squareup.retrofit2:converter-simplexml
    - JAXB: com.squareup.retrofit2:converter-jaxb
    - Scalars(primitives, boxed, and String): com.squareup.retrofit2:converter-scalars

• Retrofit2 사용 등록

```
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'
```

18-2 HTTP 통신하기

- 모델 클래스 선언
  - 모델 클래스의 프로퍼티에 데이터가 자동으로 저장되는 기본 규칙은 데이터의 키와 프로퍼티 이름을 매칭
  - 만약 키와 프로퍼티 이름이 다를 때는 @SerializedName이라는 애너테이션으로 명시

```
{
    "id": 7,
    "email": "michael.lawson@reqres.in",
    "first_name": "Michael",
    "last_name": "Lawson",
    "avatar": "https://reqres.in/img/faces/7-image.jpg"
}
```

• 모델 클래스

```
data class UserModel(
    var id: String,
    @SerializedName("first_name")
    var firstName: String,
    // @SerializedName("last_name")
    var lastName: String,
    var avatar: String,
    var avatarBitmap: Bitmap
)
```

18-2 HTTP 통신하기

- 서비스 인터페이스 정의
  - @GET은 서버와 연동할 때 GET 방식으로 해달라는 의미
  - @Query는 서버에 전달되는 데이터
  - @Url은 요청 URL

• 서비스 인터페이스 정의

```
interface INetworkService {
    @GET("api/users")
    fun doGetUserList(@Query("page") page: String): Call<UserListModel>
    @GET
    fun getAvatarImage(@Url url: String): Call<ResponseBody>
}
```

18-2 HTTP 통신하기

- Retrofit 객체 생성
  - baseUrl() 함수로 URL을 설정
  - addConverterFactory() 함수로 데이터를 파싱해 모델 객체에 담는 역할자를 지정

• Retrofit 객체 생성

```
val retrofit: Retrofit
    get() = Retrofit.Builder()
        .baseUrl("https://reqres.in/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()
```

18-2 HTTP 통신하기

- 인터페이스 타입의 서비스 객체 얻기
  - Retrofit의 create() 함수에 앞에서 만든 서비스 인터페이스 타입을 전달

• 서비스 객체 얻기

```
var networkService: INetworkService = retrofit.create(INetworkService::class.java)
```

18-2 HTTP 통신하기

- 네트워크 통신 시도
  - 터페이스의 함수를 호출하면 네트워크 통신을 시도
  - Call 객체가 반환
  - Call 객체의 enqueue() 함수를 호출하는 순간 네트워킹

• Call 객체 얻기

```
val userListCall = networkService.doGetUserList("1")
```

• 네트워크 통신 수행

```
userListCall.enqueue(object : Callback<UserListModel> {
    override fun onResponse(call: Call<UserListModel>,
                            response: Response<UserListModel>) {
        val userList = response.body()
        (... 생략 ...)
    }
    override fun onFailure(call: Call<UserListModel>, t: Throwable) {
        call.cancel()
    }
})
```

18-2 HTTP 통신하기

Retrofit 애너테이션

- @GET, @POST, @PUT, @DELETE, @HEAD
  - HTTP 메서드를 정의하는 애너테이션

- @Path
  - URL의 경로를 동적으로 지정

• HTTP 메서드 애너테이션

• 인터페이스에 선언한 함수

```
@GET("users/list?sort=desc")
fun test1(): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val call: Call<UserModel> = networkService.test1()
```

• 최종 서버 요청 URL

```
https://reqres.in/users/list?sort=desc
```

• 동적인 경로 애너테이션

• 인터페이스에 선언한 함수

```
@GET("group/{id}/users/{name}")
fun test2(
    @Path("id") userId: String,
    @Path("name") arg2: String
): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val call: Call<UserModel> = networkService.test2("10", "kkang")
```

• 최종 서버 요청 URL

```
https://reqres.in/group/10/users/kkang
```

18-2 HTTP 통신하기

- @Query
  - 함수의 매개변숫값을 서버에 전달

• 질의 애너테이션 예

• 인터페이스에 선언한 함수

```
@GET("group/users")
fun test3(
    @Query("sort") arg1: String,
    @Query("name") arg2: String
): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val call: Call<UserModel> = networkService.test3("age", "kkang")
```

• 최종 서버 요청 URL

```
https://reqres.in/group/users?sort=age&name=kkang
```

18-2 HTTP 통신하기

- @QueryMap
  - 서버에 전송할 데이터를 Map 타입의 매개변수 처리

• 질의 맵 애너테이션 예

• 인터페이스에 선언한 함수

```
@GET("group/users")
fun test4(
    @QueryMap options: Map<String, String>,
    @Query("name") name: String
): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val call: Call<UserModel> = networkService.test4(
    mapOf<String, String>("one" to "hello", "two" to "world"),
    "kkang"
)
```

• 최종 서버 요청 URL

```
https://reqres.in/group/users?one=hello&two=world&name=kkang
```

18-2 HTTP 통신하기

- @Body
  - 서버에 전송할 데이터를 모델 객체로 지정

• 모델 객체 애너테이션 예

• 인터페이스에 선언한 함수

```
@POST("group/users")
fun test5(
    @Body user: UserModel,
    @Query("name") name: String
): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val call: Call<UserModel> = networkService.test5(
    UserModel(id="1", firstName = "gildong", lastName = "hong", avatar = "someurl"),
    "kkang"
)
```

• 최종 서버 요청 URL

```
https://reqres.in/group/users?name=kkang
```

• 서버에 스트림으로 전송되는 데이터

```
{"id":"1","first_name":"gidong","last_name":"hong", "avatar":"someurl"}
```

18-2 HTTP 통신하기

- @FormUrlEncoded와 @Field
  - @FormUrlEncoded 애너테이션은 데이터를 URL 인코딩 형태로 만들어 전송
  - @Field 애너테이션이 추가된 데이터를 인코딩해서 전송

• URL 인코딩 애너테이션 예

• 인터페이스에 선언한 함수

```
@FormUrlEncoded
@POST("user/edit")
fun test6(
    @Field("first_name") first: String?,
    @Field("last_name") last: String?,
    @Query("name") name: String?
): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val call: Call<UserModel> = networkService.test6(
    "gildong 길동",
    "hong 홍",
    "kkang"
)
```

• 최종 서버 요청 URL

```
https://reqres.in/user/edit?name=kkang
```

• 서버에 스트림으로 전송되는 데이터

```
first_name=gildong%20%EA%B8%B8%EB%8F%99&last_name=hong%20%ED%99%8D
```

18-2 HTTP 통신하기

- 배열이나 List 객체에 @Field 애너테이션을 사용하면 데이터 여러 건을 같은 키로 서버에 전달

• 리스트에 필드 애너테이션 사용 예

• 인터페이스에 선언한 함수

```
@FormUrlEncoded
@POST("tasks")
fun test7(@Field("title") titles: List<String>): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val list: MutableList<String> = ArrayList()
list.add("홍길동")
list.add("류현진")
val call = networkService.test7(list)
```

• 최종 서버 요청 URL

```
https://reqres.in/tasks
```

• 서버에 스트림으로 전송되는 데이터

```
title=%ED%99%8D%EA%B8%B8%EB%8F%99&title=%EB%A5%98%ED%98%84%EC%A7%84
```

18-2 HTTP 통신하기

- @Header
  - 서버 요청에서 헤더값을 조정

• 헤더 애너테이션 예

• 인터페이스에 선언한 함수

```
@Headers("Cache-Control: max-age=640000")
@GET("widget/list")
fun test8(): Call<UserModel>
```

- @Url
  - baseUrl을 무시하고 전혀 다른 URL을 지정

• URI 애너테이션 예

• 인터페이스에 선언한 함수

```
@GET
fun test9(@Url url: String, @Query("name") name: String): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val call = networkService.test9("http://www.google.com", "kkang")
```

• 최종 서버 요청 URL

```
http://www.google.com/?name=kkang
```

18-3 이미지 처리하기 – Glide 라이브러리

- Glide는 모든 종류의 이미지를 가능한 한 빠르게 가져와서 이용
- 이미지의 크기를 조절하거나 로딩 이미지, 오류 이미지 표시 등을 쉽게 구현

• Glide 등록

```
implementation 'com.github.bumptech.glide:glide:4.12.0'
```

이미지를 가져와 출력하기

- load() 함수에 리소스를 전달하고 into() 함수에 이미지 뷰 객체를 전달

• 리소스 이미지 출력

```
Glide.with(this)
    .load(R.drawable.seoul)
    .into(binding.resultView)
```

18-3 이미지 처리하기 – Glide 라이브러리

• 파일 이미지 출력

```
val requestLauncher = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult())
{
    Glide.with(this)
        .load(it.data!!.data)
        .into(binding.resultView)
}
val intent = Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI)
intent.type = "image/*"
requestLauncher.launch(intent)
```

• 서버 이미지 출력

```
Glide.with(this)
    .load(url)
    .into(binding.resultView)
```

18-3 이미지 처리하기 – Glide 라이브러리

크기 조절

- Glide를 이용하면 코드에서 이미지 크기를 줄이지 않아도 이미지 뷰의 크기에 맞게 자동으로 줄여서 불러옵니다.
- 특정한 크기로 이미지를 줄이고 싶다면 다음처럼 override() 함수를 사용

• 크기 조절

```
Glide.with(this)
    .load(R.drawable.seoul)
    .override(200, 200)
    .into(binding.resultView)
```

18-3 이미지 처리하기 – Glide 라이브러리

로딩, 오류 이미지 출력

- placeholder(), error() 함수를 사용해 해당 상황에 표시할 이미지를 지정

• 로딩, 오류 이미지 출력

```
Glide.with(this)
    .load(url)
    .override(200, 200)
    .placeholder(R.drawable.loading)
    .error(R.drawable.error)
    .into(binding.resultView)
```

18-3 이미지 처리하기 – Glide 라이브러리

이지미 데이터 사용하기

- into() 함수에 CustomTarget 객체를 지정

• 이미지 데이터 사용하기

```
Glide.with(this)
    .load(url)
    .into(object : CustomTarget<Drawable>() {
        override fun onResourceReady(
            resource: Drawable,
            transition: Transition<in Drawable>?
        ) {
            TODO("Not yet implemented")
        }
        override fun onLoadCleared(placeholder: Drawable?) {
            TODO("Not yet implemented")
        }
    })
```

불러온 이미지 데이터

Do it! 실습

뉴스 앱 만들기

1단계. 모듈 생성하고 빌드 그래들 수정하기

- Ch18_Network 라는 이름으로 새로운 모듈을 만듭니다.
- 뷰 바인딩 기법을 이용하도록 설정
- 라이브러리를 추가

2단계. 실습 파일 복사하기

- res의 layout, menu 디렉터리를 현재 모듈의 같은 위치에 복사
- 소스 디렉터리에 있는 하위 디렉터리와 파일을 소스 영역에 모두 복사

Do it! 실습

시계 앱의 스톱워치 기능 만들기

3단계. 매니페스트 수정하기

- 네트워크 통신 관련 퍼미션과 앱의 Application 객체를 추가

4단계. 아이템과 페이지 모델 클래스 작성하기

- model 패키지의 ItemModel.kt 파일을 열고 아이템 모델 클래스를 작성

5단계. MyApplication.kt 작성하기

- MyApplication.kt 파일을 열고 작성

시계 앱의 스톱워치 기능 만들기

6단계. 리사이클러 뷰의 어댑터 작성하기

- recycler 패키지의 MyAdapter.kt 파일을 열고 작성

7단계. Retrofit 의 인터페이스 작성하기

- retrofit 패키지의 NetworkService.kt 파일을 열고 작성

8단계. 프래그먼트 작성하기

- VolleyFragment.kt 파일을 열고 작성

Do it! 실습

시계 앱의 스톱워치 기능 만들기

9단계. 앱 실행하기

감사합니다

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

윌리엄 셰익스피어
William Shakespeare