SQLite

I. 실습

1. 실습2

1) FoodDBHelper.kt

```
class FoodDBHelper(context: Context?) : SQLiteOpenHelper(context, DB_NAME, null, 1) {
    val TAG = "FoodDBHelper"

    companion object {
        const val DB_NAME = "food_db"
        const val TABLE_NAME = "food_table"
        const val COL_FOOD = "food"
        const val COL_COUNTRY = "country"
    }

    override fun onCreate(db: SQLiteDatabase?) {
        val CREATE_TABLE =
            "CREATE TABLE food_table (${BaseColumns._ID} INTEGER PRIMARY KEY AUTOINCREMENT,food TEXT, country
TEXT)"
        Log.d(TAG, CREATE_TABLE)
        db?.execSQL(CREATE_TABLE)

        db?.execSQL("INSERT INTO food_table VALUES (null, '불고기', '한국')")
        db?.execSQL("INSERT INTO food_table VALUES (null, '비빔밥', '한국')")
        db?.execSQL("INSERT INTO food_table VALUES (null, '마라탕', '중극')")
        db?.execSQL("INSERT INTO food_table VALUES (null, '딤섬', '중국')")
        db?.execSQL("INSERT INTO food_table VALUES (null, '스시', '일본')")
        db?.execSQL("INSERT INTO food_table VALUES (null, '오코노미야키', '일본')")
    }

    override fun onUpgrade(db: SQLiteDatabase?, oldVer: Int, newVer: Int) {
        val DROP_TABLE = "DROP TABLE IF EXISTS food_table"
        db?.execSQL(DROP_TABLE)
        onCreate(db)
    }
}
```

2) FoodDto.kt

```
data class FoodDto(val no: Int, var food: String, var country: String) {
    override fun toString() = "$no - $food ( $country )"
}
```

3) MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    val TAG = "MainActivity"

    lateinit var binding : ActivityMainBinding

    lateinit var helper : FoodDBHelper

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        helper = FoodDBHelper(this)
```

```
        binding.btnSelect.setOnClickListener{
            showFoods()

        }

        binding.btnAdd.setOnClickListener{
            //addFood("햄버거", "미국")
            val intent = Intent(this, AddActivity::class.java)
            startActivity(intent)

        }

        binding.btnUpdate.setOnClickListener{
            //modifyFood()
            val intent = Intent(this, UpdateActivity::class.java)
            startActivity(intent)
        }

        binding.btnRemove.setOnClickListener{
            //deleteFood()
            val intent = Intent(this, RemoveActivity::class.java)
            startActivity(intent)
        }

    }

    @SuppressLint("Range")
    fun showFoods() { // select
        val list = ArrayList<FoodDto>()
        val db = helper.readableDatabase
        val columns = null
        val selection = null
        val selectArgs =null
        val cursor = db.query("food_table", columns, selection, selectArgs, null, null,null)

        with(cursor) {
            while (moveToNext()) {
                val id = getInt(getColumnIndex(BaseColumns._ID))
                val food = getString(getColumnIndex("food"))
                val country = getString(getColumnIndex("country"))
                list.add(FoodDto(id, food, country))
            }
            close()
        }

        helper.close()

        var data = ""

        for (dto in list) {
            data += dto.toString() + "\n"
        }

        binding.tvDisplay.text=data // Update the text of the TextView with the string 'data'
    }
}
```

4) AddActivity.kt

```
class AddActivity : AppCompatActivity() {

    lateinit var addBinding : ActivityAddBinding
    lateinit var helper : FoodDBHelper

    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
        addBinding = ActivityAddBinding.inflate(layoutInflater)
        setContentView(addBinding.root)

        helper = FoodDBHelper(this)  // 'var' 키워드 제거

        val AddFood = addBinding.etAddFood
        val AddCountry = addBinding.etAddNation
        val btnAdd = addBinding.btnAddFood
        val btnCancel = addBinding.btnAddCancel

        btnAdd.setOnClickListener {
            val food = AddFood.text.toString()
            val country = AddCountry.text.toString()
            addFood(food, country)  // 함수 호출 추가
            finish()  // 데이터 추가 후 액티비티 종료
        }

        btnCancel.setOnClickListener {
            finish()  // 취소 버튼 클릭시 액티비티 종료
        }
    }

    fun addFood(food: String, country: String) {
        val db = helper.writableDatabase
        val newRow = ContentValues()
        newRow.put("food", food)
        newRow.put("country", country)
        db.insert("food_table", null, newRow)

        db.close()  // 수정된 부분 (helper 대신에 db.close() 호출)

    }
}


//추가 버튼 클릭시 -> 데이터베이스에 반영
//취소 버튼 클릭시 -> 데이터베이스에 미반영
```

5) RemoveActivity.kt

```
class RemoveActivity : AppCompatActivity() {

    lateinit var helper : FoodDBHelper

    lateinit var removeBinding : ActivityRemoveBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        removeBinding = ActivityRemoveBinding.inflate(layoutInflater)
        setContentView(removeBinding.root)

        helper = FoodDBHelper(this)

        val reFood = removeBinding.etRemoveFood
        val btnUp = removeBinding.btnRemoveFood
        val btnCancel = removeBinding.btnRemoveCancel

        btnUp.setOnClickListener {
            val food = reFood.text.toString()
            deleteFood(food)
            finish()  // 데이터 수정 후 액티비티 종료
        }

        btnCancel.setOnClickListener {
```

```
            finish()  // 취소 버튼 클릭시 액티비티 종료
        }
    }

    fun deleteFood(food: String) {
        val db=helper.writableDatabase

        val whereClause="food=?"
        val whereArgs= arrayOf(food)

        db.delete("food_table" ,whereClause ,whereArgs )

        helper.close()
    }
}
```

6) UpdateActivity.kt

```
class UpdateActivity : AppCompatActivity() {

    lateinit var updateBinding: ActivityUpdateBinding
    lateinit var helper: FoodDBHelper

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        updateBinding = ActivityUpdateBinding.inflate(layoutInflater)
        setContentView(updateBinding.root)

        helper = FoodDBHelper(this)

        val upId = updateBinding.etUpdateId
        val upFood = updateBinding.etUpdateFood
        val btnUp = updateBinding.btnUpdateFood
        val btnCancel = updateBinding.btnUpdateCancel

        btnUp.setOnClickListener {
            val id = upId.text.toString()
            val food = upFood.text.toString()
            modifyFood(id.toInt(), food)
            finish()  // 데이터 수정 후 액티비티 종료
        }

        btnCancel.setOnClickListener {
            finish()  // 취소 버튼 클릭시 액티비티 종료
        }
    }

    private fun modifyFood(id: Int, food: String) {
        val db = helper.writableDatabase
        val updateRow = ContentValues()

        updateRow.put("food", food)

        val whereClause="${BaseColumns._ID}=?"
        val whereArgs= arrayOf(id.toString())

        db.update("food_table" ,updateRow ,whereClause ,whereArgs )

        db.close()
    }
}
```

Room 활용

I. 실습

1. 실습2

1) Food.kt

```
@Entity
(tableName = "food_table2")
data class Food(
    @PrimaryKey(autoGenerate = true)
    var _id: Int? = null,
    var food: String,
    var country: String
) {
    override fun toString(): String {
        return "$_id - $food ($country)"
    }
}
```

2) FoodDao.kt

```
@Dao
interface FoodDao {
    @Query("SELECT * FROM food_table2")
    fun getAllFoods() : Flow<List<Food>>

    @Query("SELECT * FROM food_table2 WHERE country = :country")
    suspend fun getFoodByCountry(country: String) : List<Food>

    @Insert
    suspend fun insertFood(vararg food : Food)

    @Query("UPDATE food_table2 SET country = :country WHERE food = :foodName")
    suspend fun updateFood(foodName: String, country: String)

    @Query("DELETE FROM food_table2 WHERE food = :foodName")
    suspend fun deleteFood(foodName: String)
}
```

3) FoodDatabase.kt

```
@Database(entities = [Food::class], version=1)
abstract class FoodDatabase : RoomDatabase() {
    abstract fun foodDao() : FoodDao

    companion object {
        @Volatile      // Main memory 에 저장한 값 사용
        private var INSTANCE : FoodDatabase? = null

        // INSTANCE 가 null 이 아니면 반환, null 이면 생성하여 반환
        fun getDatabase(context: Context) : FoodDatabase {
            return INSTANCE ?: synchronized(this) {     // 단일 스레드만 접근
                val instance = Room.databaseBuilder(context.applicationContext,
                    FoodDatabase::class.java, "food_db").build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

4) FoodAdapter.kt

```
class FoodAdapter(val foods: List<Food>) : RecyclerView.Adapter<FoodAdapter.FoodViewHolder>() {
    val TAG = "FoodAdapter"

    override fun getItemCount(): Int {
        return foods.size
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): FoodViewHolder {
        val itemBinding = ListItemBinding.inflate(LayoutInflater.from(parent.context), parent, false)
        return FoodViewHolder(itemBinding)
    }

    override fun onBindViewHolder(holder: FoodViewHolder, position: Int) {
        holder.itemBinding.tvItem.text = foods[position].toString()
        holder.itemBinding.root.setOnLongClickListener{
            itemLongClickListener?.onItemLongClickListener(it, position)
            true
        }
    }

    class FoodViewHolder(val itemBinding: ListItemBinding)
        : RecyclerView.ViewHolder(itemBinding.root)


    interface OnItemLongClickListener {
        fun onItemLongClickListener(view: View, pos: Int)
    }

    var itemLongClickListener : OnItemLongClickListener? = null

    fun setOnItemLongClickListener(listener: OnItemLongClickListener?) {
        itemLongClickListener = listener
    }

}
```

5) MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    val TAG = "MainActivity"
    lateinit var binding: ActivityMainBinding
    lateinit var db : FoodDatabase // DB 인스턴스
    lateinit var foodDao : FoodDao // DAO 인스턴스

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        db = FoodDatabase.getDatabase(this)
        foodDao = db.foodDao()

        /*샘플 데이터, DB 사용 시 DB에서 읽어온 데이터로 대체 필요*/
        val foods = ArrayList<Food>()
        foods.add(Food(1,"된장찌개", "한국"))
        foods.add(Food(2,"김치찌개", "한국"))
        foods.add(Food(3,"마라탕", "중국"))
        foods.add(Food(4,"훠궈", "중국"))
        foods.add(Food(5,"스시", "일본"))
        foods.add(Food(6,"오코노미야키", "일본"))

        CoroutineScope(IO).launch {
            foods.forEach { food ->
```

```
            foodDao.insertFood(food)
        }
        showAllFoods()
    }

    binding.btnShow.setOnClickListener{
        val country = binding.etNation.text.toString()
        showFoodByCountry(country)
    } // 입력한 나라만 보이도록
    binding.btnInsert.setOnClickListener{
        val food2 = binding.etFood.text.toString()
        val country2 = binding.etNation.text.toString()
        addFood(Food(food=food2, country=country2))
    } // 음식 및 나라 입력

    binding.btnUpdate.setOnClickListener {
        val name = binding.etFood.text.toString()
        val Country2 = binding.etNation.text.toString()
        modify(name, Country2)
    } // 음식 기준 나라이름 변경

    binding.btnDelete.setOnClickListener {
        val food = binding.etFood.text.toString()
        remove(food)
    } // 음식 기준 삭제
}

fun showAllFoods() {
    CoroutineScope(Dispatchers.IO).launch {
        val flowFoods = foodDao.getAllFoods()
        flowFoods.collect{foods ->
            for (food in foods) {
                Log.d(TAG, food.toString())
            }
        }
    }
}

fun showFoodByCountry(country: String) {
    CoroutineScope(Dispatchers.IO).launch {
        val foods = foodDao.getFoodByCountry(country)
        for (food in foods) {
            Log.d(TAG, food.toString())
        }
    }
}

 fun addFood(food: Food) {
    CoroutineScope(IO).launch {
        foodDao.insertFood(food)
    }
}

fun modify(foodName: String, newCountry: String) {
    CoroutineScope(IO).launch {
        foodDao.updateFood(foodName, newCountry)
    }
}

fun remove(foodName: String) {
    CoroutineScope(IO).launch {
        foodDao.deleteFood(foodName)
    }
}
}
```

HTTP Communication

I. 실습

1. 실습2

1) NetworkManager.kt

```
class NetworkManager(val context: Context) {
    private val TAG = "NetworkManager"

    fun downloadText(url: String) : String? {
        var receivedContents : String? = null
        var iStream : InputStream? = null
        var conn : HttpURLConnection? = null
//        var conn : HttpsURLConnection? = null

        try {
            val url : URL = URL(url)
            conn = url.openConnection() as HttpURLConnection      // 서버 연결 설정 - MalformedURLException
            // conn = url.openConnection() as HttpsURLConnection      // 서버 연결 설정 - MalformedURLException

            conn.readTimeout = 5000                              // 읽기 타임아웃 지정 - SocketTimeoutException
            conn.connectTimeout = 5000                           // 연결 타임아웃 지정 - SocketTimeoutException
            conn.doInput = true                                  // 서버 응답 지정 - default
            conn.requestMethod = "GET"                           // 연결 방식 지정 - or POST
            conn.setRequestProperty("content-type", "application/x-www-form-urlencoded;charset=EUC-KR")
            // 전송 형식 지정  json 일 경우 application/json 으로 변경

//            conn.connect()                                     // 통신 링크 열기 - 트래픽 발생
            val responseCode = conn.responseCode                 // 서버 전송 및 응답 결과 수신

            if (responseCode != HttpsURLConnection.HTTP_OK) {
                throw IOException("Http Error Code: $responseCode")
            }

            iStream = conn.inputStream                          // 응답 결과 스트림 확인
            receivedContents = readStreamToString (iStream)       // stream 처리 함수를 구현한 후 사용

        } catch (e: Exception) {       // MalformedURLException, IOExceptionl, SocketTimeoutException 등 처리 필요
            Log.d(TAG, e.message!!)
        } finally {
            if (iStream != null) { try { iStream.close()} catch (e: IOException) { Log.d(TAG, e.message!!) } }
            if (conn != null) conn.disconnect()
        }
        return receivedContents
    }

    fun downloadImage(url : String) : Bitmap? {
        var receivedContents : Bitmap? = null
        var iStream : InputStream? = null
        var conn : HttpURLConnection? = null
        // Image Download 구현
        try {
            val url : URL = URL(url)
            conn = url.openConnection() as HttpURLConnection      // 서버 연결 설정

            conn.readTimeout = 5000                              // 읽기 타임아웃 지정
            conn.connectTimeout = 5000                           // 연결 타임아웃 지정
            conn.doInput = true                                  // 서버 응답 지정 - default
            conn.requestMethod = "GET"                           // 연결 방식 지정 - or POST

            val responseCode = conn.responseCode                 // 서버 전송 및 응답 결과 수신
```

```
        if (responseCode != HttpURLConnection.HTTP_OK) {
            throw IOException("Http Error Code: $responseCode")
        }

        iStream = conn.inputStream                      // 응답 결과 스트림 확인
        receivedContents = BitmapFactory.decodeStream(iStream)   // InputStream을 Bitmap으로 변환

    } catch (e: Exception) {       // MalformedURLException, IOExceptionl, SocketTimeoutException 등 처리 필요
        Log.d(TAG, e.message!!)
    } finally {
        if (iStream != null) { try { iStream.close()} catch (e: IOException) { Log.d(TAG, e.message!!) } }
        if (conn != null) conn.disconnect()
    }
    return receivedContents
}

fun sendPostData(url: String) : String? {
    var receivedContents : String? = null
    var iStream : InputStream? = null
    var conn : HttpsURLConnection? = null

    // POST 요청 구현
    try {
        val url : URL = URL(url)
        conn = url.openConnection() as HttpsURLConnection      // 서버 연결 설정

        conn.readTimeout = 5000                             // 읽기 타임아웃 지정
        conn.connectTimeout = 5000                          // 연결 타임아웃 지정
        conn.doInput = true                                 // 서버 응답 지정 - default
        conn.doOutput = true                                // 서버에 데이터를 전송할 것임을 지정
        conn.requestMethod = "POST"                         // 연결 방식 지정 - POST
        conn.setRequestProperty("content-type","application/x-www-form-urlencode; charset=UTF-8")

        // POST 요청 본문에 들어갈 파라미터를 설정
        val postData = StringBuilder()
        for ((key, value) in params) {
            if (postData.isNotEmpty()) postData.append('&')
            postData.append(URLEncoder.encode(key, "UTF-8"))
            postData.append('=')
            postData.append(URLEncoder.encode(value, "UTF-8"))
        }

//          val params1 = "&itemPerPage=$itemPerPage"
//          val params2 = "&multiMovieYn=$multiMovieYn"
        val writer = BufferedWriter(OutputStreamWriter(conn.outputStream, "UTF-8"))
//          writer.write(params1)
//          writer.write(params1)   // 서버로 전송할 데이터 쓰기
        writer.write(postData.toString()) // 서버로 전송할 데이터 쓰기
        writer.flush()
        writer.close()

        val responseCode = conn.responseCode                // 서버 전송 및 응답 결과 수신

        if (responseCode != HttpsURLConnection.HTTP_OK) {
            throw IOException("Http Error Code: $responseCode")
        }

        iStream = conn.inputStream                          // 응답 결과 스트림 확인
        receivedContents = readStreamToString(iStream)         // InputStream을 문자열로 변환

    } catch (e: Exception) {       // MalformedURLException, IOException, SocketTimeoutException 등 처리 필요
        Log.d(TAG, e.message!!)
    } finally {
        if (iStream != null) { try { iStream.close()} catch (e: IOException) { Log.d(TAG, e.message!!) } }
```

```
            if (conn != null) conn.disconnect()
        }
        return receivedContents
    }

    // InputStream 을 String 으로 변환
    private fun readStreamToString(iStream : InputStream?) : String {
        val resultBuilder = StringBuilder()

        val inputStreamReader = InputStreamReader(iStream)
        val bufferedReader = BufferedReader(inputStreamReader)

        var readLine : String? = bufferedReader.readLine()
        while (readLine != null) {
            resultBuilder.append(readLine + System.lineSeparator())
            readLine = bufferedReader.readLine()
        }

        bufferedReader.close()
        return resultBuilder.toString()
    }

    // InputStream 을 Bitmap 으로 변환
    private fun readStreamToImage(iStream: InputStream?) : Bitmap {
        val bitmap = BitmapFactory.decodeStream(iStream)
        return bitmap
    }
}
```

2) MainActivity.kt

```
class MainActivity : AppCompatActivity() {
    val TAG = "MainActivity"

    lateinit var binding: ActivityMainBinding
    lateinit var networkManager : NetworkManager

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        networkManager = NetworkManager(this)


        val pol = StrictMode.ThreadPolicy.Builder().permitNetwork()
            .build() // 임시로 걍 main_thread 사용할 수 있도록(원래 안됨!)
        StrictMode.setThreadPolicy(pol)


        binding.btnConnInfo.setOnClickListener {
            getNetworkInfo()
        }

        binding.btnActiveInfo.setOnClickListener {
            Log.d(TAG, "Network is connected: ${isOnline()}")
        }

        binding.btnDown.setOnClickListener {
            // val url = "https://httpbin.org/get?user=somsom&dept=computer"
            val url = resources.getString(R.string.kobis_url) + "20231001"
            binding.tvDisplay.text = networkManager.downloadText(url)
        }

        binding.btnImg.setOnClickListener {
```

```
            val imageUrl = resources.getString(R.string.image_url)
            val result = networkManager.downloadImage(imageUrl)
            binding.ivDisplay.setImageBitmap(result)
        }

        binding.btnSend.setOnClickListener {
            //binding.tvDisplay.text = networkManager.sendPostData("https://httpbin.org/post")
            val params = HashMap<String, String>()
            params["itemPerPage"] = "10"
            params["multiMovieYn"] = "N"
            binding.tvDisplay.text = networkManager.sendPostData("https://httpbin.org/post", params)
        }
    }

    private fun getNetworkInfo() {
        val connMgr = getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
        var isWifiConn: Boolean = false
        var isMobileConn: Boolean = false

        connMgr.allNetworks.forEach { network ->
            connMgr.getNetworkInfo(network)?.apply {
                if (type == ConnectivityManager.TYPE_WIFI) {
                    isWifiConn = isWifiConn or isConnected
                }
                if (type == ConnectivityManager.TYPE_MOBILE) {
                    isMobileConn = isMobileConn or isConnected
                }
            }
        }

        Log.d(TAG, "Wifi connected: $isWifiConn")
        Log.d(TAG, "Mobile connected: $isMobileConn")
    }

    private fun isOnline(): Boolean { // 최소한 얘는 되어 있어야 함 <필수>
        val connMgr = getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
        val networkInfo: NetworkInfo? = connMgr.activeNetworkInfo
        return networkInfo?.isConnected == true
    }
}
```

네트워크 기능 구현(XmlPullParser)

I. OpenAPI

1. 과제(실습)

1) Book.kt

```
data class Book(
    var title: String?,
    var author: String?,
    var publisher: String?
)
```

2) NaverBookParser.kt

```
class NaverBookParser {
    private val ns: String? = null

    companion object {
        val FAULT_RESULT = "faultResult"
        val CHANNEL_TAG = "channel"
        val TITLE_TAG = "title"
        val AUTHOR_TAG = "author"
        val PUBLISHER_TAG = "publisher"
    }

    @Throws(XmlPullParserException::class, IOException::class)
    fun parse(inputStream: InputStream?) : List<Book> {

        inputStream.use { inputStream ->
            val parser : XmlPullParser = Xml.newPullParser()

            /*Parser 의 동작 정의, next() 호출 전 반드시 호출 필요*/
            parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false)

            /* Paring 대상이 되는 inputStream 설정 */
            parser.setInput(inputStream, null)

            /*Parsing 대상 태그의 상위 태그까지 이동*/
            while(parser.name != "channel") {
                parser.next()
            }
            return readBookList(parser)
        }
    }

    @Throws(XmlPullParserException::class, IOException::class)
    private fun readBookList(parser: XmlPullParser) : List<Book> {
        val books = mutableListOf<Book>()

        parser.require(XmlPullParser.START_TAG, ns, "channel")
        while(parser.next() != XmlPullParser.END_TAG) {
            if (parser.eventType != XmlPullParser.START_TAG) {
                continue
            }
            if (parser.name == "item") { // ③ 맞는 항목의 태그인지 검사하고
                books.add( readBookItem(parser) ) // 내부에 있는 TAG를 파싱하는 함수 호출
            } else {
                skip(parser)
            }
        }
        return books
```

```
    }

    @Throws(XmlPullParserException::class, IOException::class)
    private fun readBookItem(parser: XmlPullParser) : Book {
        parser.require(XmlPullParser.START_TAG, ns, "item")
        var title : String? = null
        var author : String? = null
        var publisher : String? = null

        while (parser.next() != XmlPullParser.END_TAG) {
            if (parser.eventType != XmlPullParser.START_TAG) {
                continue
            }
            when (parser.name) { // ④ 내가 관심있는 TAG를 만나면 값을 읽는 작업
                TITLE_TAG -> title = readTextInTag(parser, TITLE_TAG) // 찾으면 또 세부에서 파싱
                AUTHOR_TAG -> author = readTextInTag(parser, AUTHOR_TAG)
                PUBLISHER_TAG -> publisher = readTextInTag(parser, PUBLISHER_TAG)
                else -> skip(parser)
            }
        }
        return Book(title ?: "", author ?: "", publisher ?: "")
    }

    @Throws(IOException::class, XmlPullParserException::class)
    private fun readTextInTag (parser: XmlPullParser, tag: String): String {
        parser.require(XmlPullParser.START_TAG, ns, tag)
        var text = ""
        if (parser.next() == XmlPullParser.TEXT) {
            text = parser.text
            parser.nextTag()
        }
        parser.require(XmlPullParser.END_TAG, ns, tag)
        return text
    }

    @Throws(XmlPullParserException::class, IOException::class)
    private fun skip(parser: XmlPullParser) {
        if (parser.eventType != XmlPullParser.START_TAG) {
            throw IllegalStateException()
        }
        var depth = 1
        while (depth != 0) {
            when (parser.next()) {
                XmlPullParser.END_TAG -> depth--
                XmlPullParser.START_TAG -> depth++
            }
        }
    }
}
```

3) NetworkManager.kt

```
class NetworkManager(val context: Context) {
    private val TAG = "NetworkManager"

    val openApiUrl by lazy {
        /* Resource 의 strings.xml 에서 필요한 정보를 읽어옴 */
        context.resources.getString(R.string.naver_url)
    }

    @Throws(IOException::class)
    fun downloadXml(keyword: String) : List<Book>? {
        var movies : List<Book>? = null

        val inputStream = downloadUrl( openApiUrl + keyword)
```

```
        /*Parser 생성 및 parsing 수행*/
        val parser = NaverBookParser()
        movies = parser.parse(inputStream)

        return movies
    }

    @Throws(IOException::class)
    private fun downloadUrl(urlString: String) : InputStream? {
        val url = URL(urlString)

        /* Resource 의 strings.xml 에서 필요한 정보를 읽어옴 */
        val cliedId = context.resources.getString(R.string.client_id)
        val clientSecret = context.resources.getString(R.string.client_secret)

        return (url.openConnection() as? HttpURLConnection)?.run {
            readTimeout = 5000
            connectTimeout = 5000
            requestMethod = "GET"
            doInput = true

            /*Naver ClientID/Secret 을 HTTP Header Property에 저장*/
            setRequestProperty("X-Naver-Client-Id", cliedId)
            setRequestProperty("X-Naver-Client-Secret", clientSecret)

            connect()
            inputStream
        }
    }

    // InputStream 을 String 으로 변환
    private fun readStreamToString(iStream : InputStream?) : String {
        val resultBuilder = StringBuilder()

        val inputStreamReader = InputStreamReader(iStream)
        val bufferedReader = BufferedReader(inputStreamReader)

        var readLine : String? = bufferedReader.readLine()
        while (readLine != null) {
            resultBuilder.append(readLine + System.lineSeparator())
            readLine = bufferedReader.readLine()
        }

        bufferedReader.close()
        return resultBuilder.toString()
    }

    // InputStream 을 Bitmap 으로 변환
    private fun readStreamToImage(iStream: InputStream?) : Bitmap {
        val bitmap = BitmapFactory.decodeStream(iStream)
        return bitmap
    }
}
```

4) MainActivity.kt

```
class MainActivity : AppCompatActivity() {
    private val TAG = "MainActivity"

    lateinit var mainBinding : ActivityMainBinding
    lateinit var adapter : MovieAdapter
    lateinit var networkDao : NetworkManager

    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
        mainBinding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(mainBinding.root)

        networkDao = NetworkManager(this)

        adapter = MovieAdapter()
        mainBinding.rvMovies.adapter = adapter
        mainBinding.rvMovies.layoutManager = LinearLayoutManager(this)

        mainBinding.btnSearch.setOnClickListener {
            val keyword = mainBinding.etKeyword.text.toString()

            CoroutineScope(Dispatchers.Main).launch{
                val def = async(Dispatchers.IO) {
                    var books : List<Book>? = null
                    try {
                        books = networkDao.downloadXml(keyword)
                    } catch (e: IOException) {
                        Log.d(TAG, e.message?: "null")
                        null
                    } catch (e: XmlPullParserException) {
                        Log.d(TAG, e.message?: "null")
                        null
                    }
                    books
                }

                adapter.books = def.await()
                adapter.notifyDataSetChanged()
            }
        }

    }
}
```