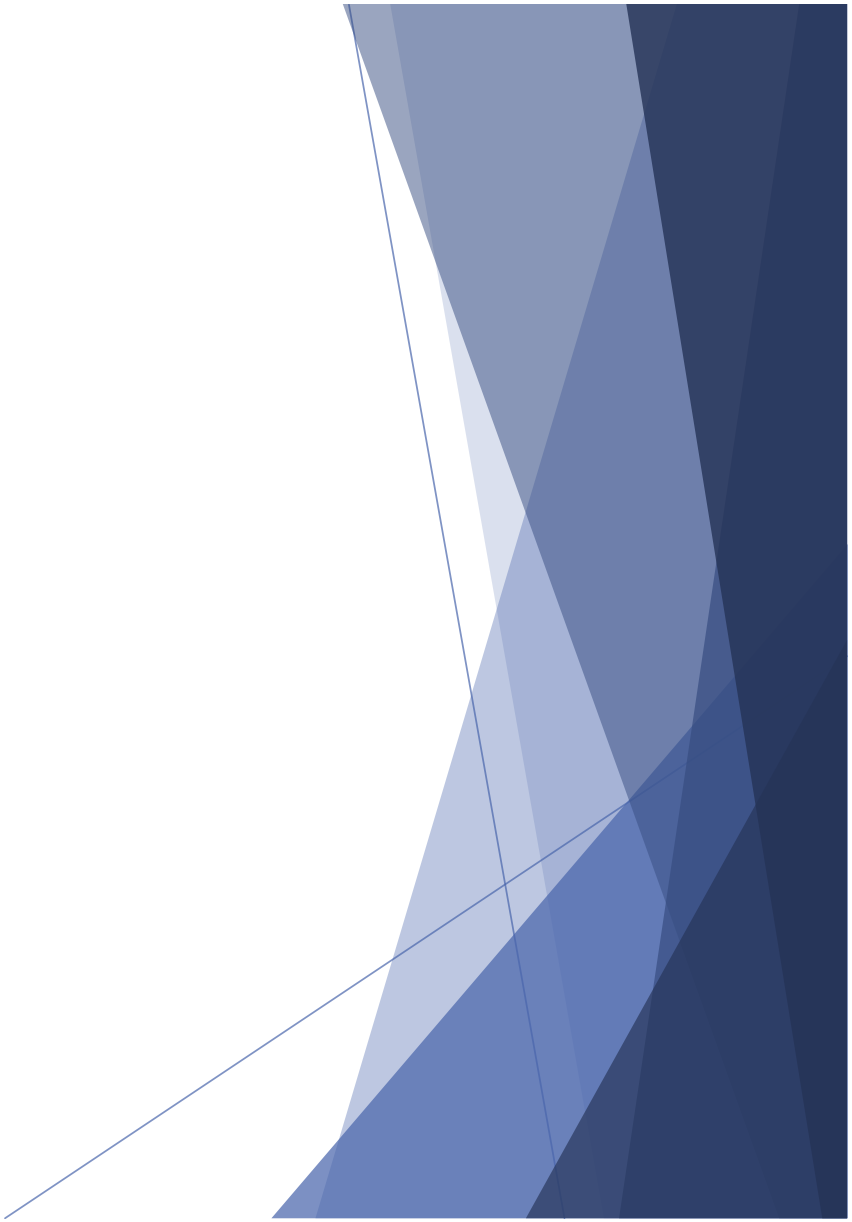
Android Programming

Thread, AsyncTask and Network

Thread와 AsyncTask



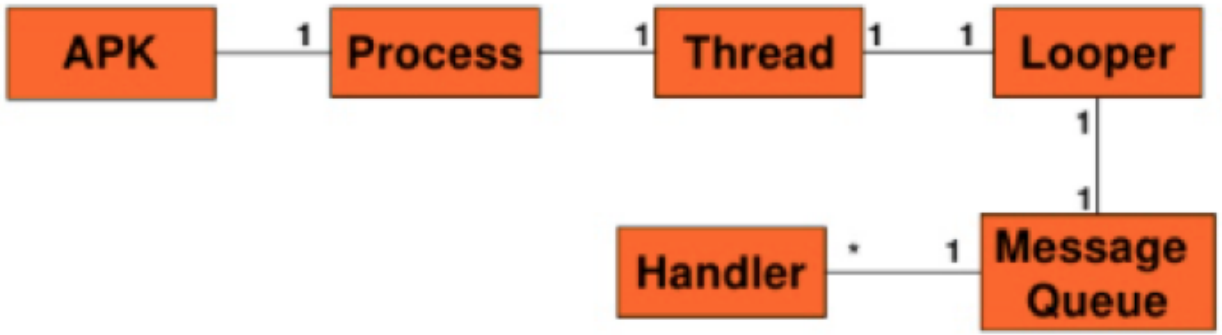
Thread

- Thread : 실타래. 하나의 작업 흐름
- 기본적으로 같은 앱 내의 모든 구성 요소는 같은 프로세스와 쓰레드에서 실행
  - 처음 앱이 시작되면 시스템이 '기본' 스레드를 생성하여 사용
  - 이 스레드는 적절한 사용자 인터페이스 위젯에 이벤트를 발송하는 역할을 주로 맡음UI 스레드라 불리기도 함
- 앱이 사용자 상호작용에 응답하여 리소스를 많이 소모하는 작업을 수행하는 경우 단일 스레드 모델은 낮은 성능을 보일 수 있음
  - 사용자 입장에서는 앱이 중단된 듯 보이며
  - UI 스레드가 5초 이상 차단되면 "애플리케이션이 응답하지 않습니다"(ANR)를 내며 종료
- 네트워크 액세스, 데이터베이스 쿼리 등 긴 작업을 수행할 때 이 부분을 회피해야 할 필요

Thread

: Android Thread 개요

- 각 프로세스는 (기본값으로) 하나의 스레드를 가짐(Single Thread Model)
- 각 스레드는 메시지 큐를 제어하기 위한 Looper를 갖고 있음
- 모든 컴포넌트의 이벤트들은 looper의 Queue에 적재되어 순차적으로 처리



Thread

: Thread의 작성

- UI Thread의 작업 흐름에 방해되지 않도록, 시간이 오래 걸리거나 리소스를 많이 차지하는 작업은 별도의 Thread로 진행해야 할 필요가 있음

- Thread의 작성법
  - Thread를 상속받아 작성
  - Runnable 인터페이스를 구현하여 작성

- Thread의 작성은 Java의 Thread API를 그대로 사용

Thread

: Thread의 작성

- Thread를 상속받는 코드의 예제
  - run 메서드를 오버라이딩 하여 수행 코드를 넣음

```
class MyThread extends Thread {
    @Override
    public void run() {
        //  Thread 수행 코드
    }
}
```

- Runnable 인터페이스를 구현하는 코드의 예제

```
class RunnableThread implements Runnable {
    @Override
    public void run() {
        //  Thread 수행 코드
    }
}
```

Thread

: Thread의 실행

- Thread를 상속받은 클래스의 실행
  - new 키워드로 Thread를 인스턴스화 한 후 start() 메서드로 실행

```
MyThread thread = new MyThread();
thread.start();
```

- Runnable 인터페이스를 구현한 클래스의 실행
  - Thread 생성자에 Runnable 인터페이스 구현 클래스를 넣음

```
Thread runnableThread = new Thread(new RunnableThread());
runnableThread.start();
```

Thread

: Thread-Handler 구조

- 남은 문제: 원칙적으로는 개발자 Thread에서 UI Thread의 구성요소에 접근할 수 없음
  - 이럴 경우, UI Thread에게 작업을 의뢰하기 위해 Handler 클래스의 post나 sendMessage를 이용

```
//  Handler의 선언
Handler handler = new Handler();
```

```
//  Handler 클래스의 선언
class UIUpdate implements Runnable {
    @Override
    public void run() {
        textView.setText("sum = " + sum);
    }
}
```

```
//  개발자 Thread에서 handler에게 UI 작업 의뢰: post 메서드의 이용
handler.post(new UIUpdate());
```

Thread

: Handler

- Handler의 sendMessage 관련 메서드
  - Message 객체를 이용, UI Thread에 넘길 데이터를 담아 전송할 수 있음

메서드	설명
`sendMessage(Message obj)`	UI Thread에 의뢰
`sendMessageAtFrontOfQueue(Message obj)`	UI Thread에게 가장 먼저 처리하라고 요청
`sendMessageAtTime(Message obj, long uptileMillis)`	지정한 시간에 의뢰를 수행해 달라고 요청
`sendMessageDelayed(Message obj, long delayMillis)`	지정 시간이 지난 후 의뢰를 수행해 달라고 요청
`sendEmptyMessage(int what)`	매개변수(데이터) 없이 작업만 의뢰

Thread

: Handler

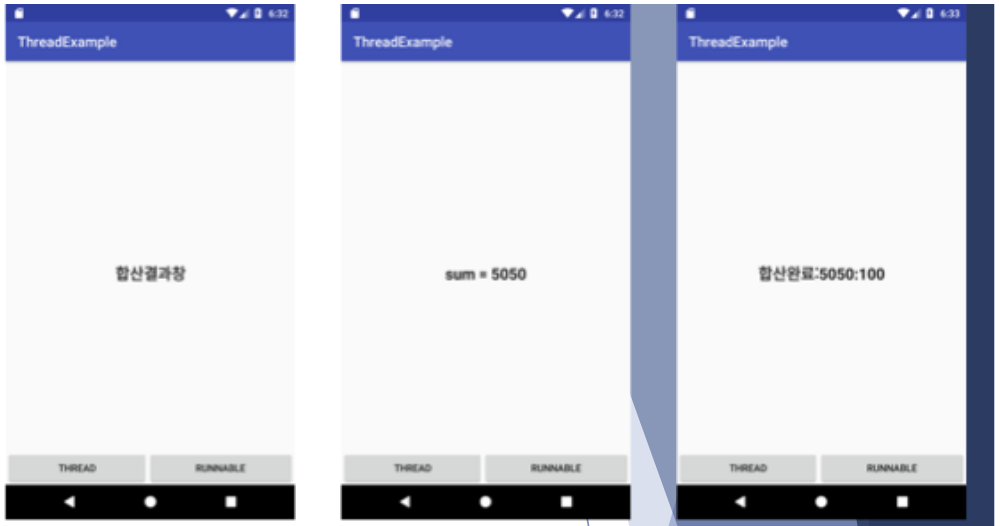
- Message 객체의 멤버 변수
  - what : int 형 변수. 작업 구분자. 요청을 구분하기 위한 임의의 숫자
  - obj : UIThread에 넘길 데이터. Object 타입
  - arg1, arg2 : UIThread에 넘길 int 형 데이터. 간단한 숫자는 arg1, arg2로 전송
- sendMessage로 전송한 메시지를 받으려면 Handler 클래스의 서브 클래스를 작성해야 함: handleMessage 메서드 오버라이드

```
handler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                if (msg.what == 1) {

                } else if (msg.what == 2) {

                }
            }
        };
```

Thread



[실습] ThreadExample

1) 프로젝트를 만들고 activity_main.xml을 교체합니다.

2) onBtnThread 메서드를 구현하고 다음과 같은 작업을 수행합니다.

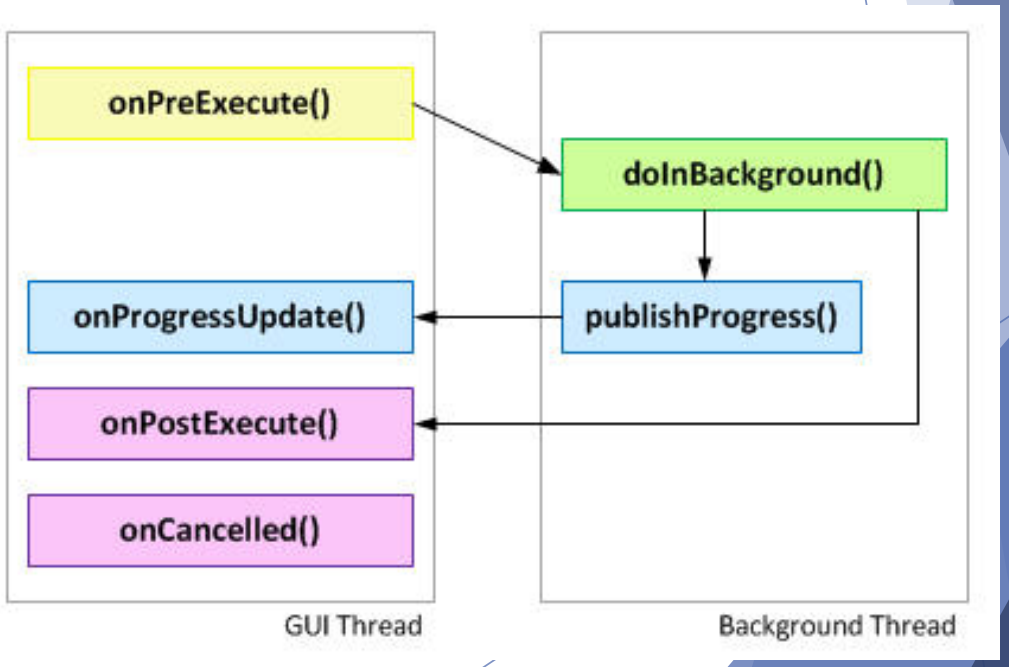
- Thread로 1 ~ 100까지 합산(각 단계별 200ms 휴식)
- 합산 결과가 변경될 때마다 Handler와 UIThread를 이용하여 textView에 합산 결과를 출력

3) onBtnRunnable 메서드를 구현하고 다음과 같은 작업을 수행합니다.

- Runnable Thread로 1~100까지 합산
- Handler에 handleMessage 메서드를 오버라이드 하고
- 합산 중일 때는 “합산중:{합산값}” 출력
- 합산 완료시 “합산완료:{합산값}:{총 개수}” 출력

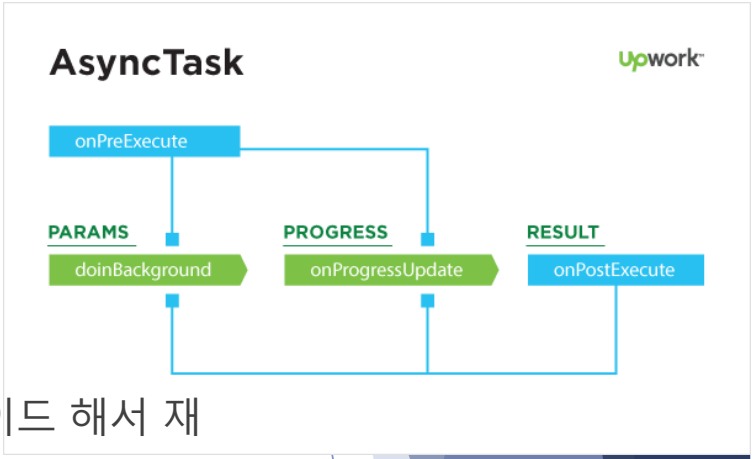
AsyncTask

- 조금 다른 방법으로 Thread-Handler 구조를 만들어 낼 수 있는 방법
  - 새로운 방법이라기 보다 Thread-Handler의 추상화 개념 정도로 이해
- AsyncTask의 흐름도



AsyncTask

: AsyncTask의 구현



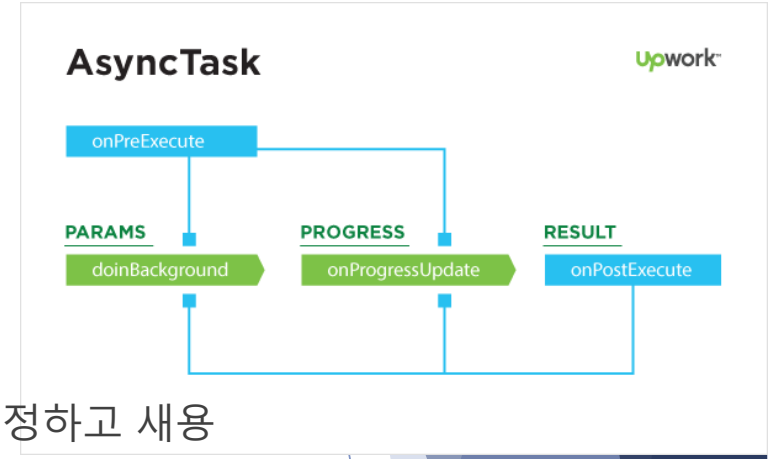
- AsyncTask는 추상 클래스이며 doInBackground 메서드는 꼭 오버라이드 해서 재정의하여야 함

메서드 명	설명
`doInBackground(Params … params)`	Thread에 처리될 내용 담는 메서드
`onPreExecute()`	AsyncTask 시작 전 가장 먼저 호출
`onPostExecute(Result result)`	작업 완료 후 가장 마지막에 호출
`onProgressUpdate(Progress … values)`	doInBackground 메서드 수행 도중 중간에 값을 받아 처리하기 위해 호출

- doInBackground 수행 도중, 중간 결과를 보내고자 한다면 publishProgress 메서드를 이용

AsyncTask

: AsyncTask의 구현



- AsyncTask 클래스 상속시에는 제네릭을 사용하여 타입을 명확히 지정하고 새용하여야 함

```
class MyAsyncTask extends AsyncTask<Void, Integer, String> {

}
```

- 각 제네릭 타입의 의미
  - 첫번째 타입: doInBackground 메서드의 매개변수 타입. 넘길 매개변수가 없으면 Void
  - 두번째 타입: doInBackground 메서드에서 publishProgress 메서드를 사용할 때 넘길 데이터 타입. 이 타입은 onProgressUpdate 메서드가 수신
  - 세번째 타입: onPostExecute 메서드의 매개변수 타입. doInBackground 메서드의 리턴값

AsyncTask

: AsyncTask의 구현

- 각 제네릭 타입의 반영 위치 참고

```
1  MyAsyncTask mProcessTask = new MyAsyncTask();
2  mProcessTask.execute( 20 );
3
4  public class MyAsyncTask extends AsyncTask< Integer , Integer , Integer > {
5      @Override
6      protected void onPreExecute() {
7          super.onPreExecute();
8      }
9
10     @Override
11     protected Integer doInBackground( Integer... integers ){
12         publishProgress( 50 );
13         return 0;
14     }
15
16     @Override
17     protected void onProgressUpdate( Integer... params ) { }
18
19     @Override
20     protected void onPostExecute( Integer result ) {
21         super.onPostExecute( result );
22     }
23 }
```

Android Network Programming

HTTP Communication

실습을 위한 서버의 구성

[실습] SimpleAPIServer

1) https://nodejs.org/ 웹사이트에 접속, 운영체제와 플랫폼에 맞는 인스톨러를 다운, 설치합니다.

2) SimpleAPIServer.zip을 다운로드 받아 압축을 풀고 접근이 쉬운 디렉터리에 위치시킵니다.

- 예) D:₩SimpleAPIServer

3) 윈도의 경우, Win + R > cmd를 입력, 다음과 같이 입력합니다

```
> D:
> cd SimpleAPIServer
> npm install
> npm start
```

4) 웹브라우저를 열고, 주소창에 다음 주소를 입력합니다.

- http://localhost:3000

HTTP 통신

- 네트워크 프로그래밍의 목적: 앱과 서버 프로그램간의 데이터 송수신
- Android는 Java의 HTTP API를 사용하여 웹 서버와 데이터를 주고 받음

- HTTP 통신 프로그램이 이용하는 Java의 패키지들
  - java.net : URL 표현 및 서버 연결에 관련된 클래스들의 모음
  - java.io : 연결된 서버와 데이터를 주고 받는 클래스들의 모음

- 안드로이드에서 인터넷을 이용하려면 INTERNET permission을 획득해야 함

```
<!-- in AndroidManifest.xml -->
<uses-permission android:name="android.permission.INTERNET" />
```

HTTP 통신

: 서버 접속

- URL 클래스 : 서버의 URL 정보를 표현
- HttpURLConnection 클래스 : URL 클래스로부터 connection을 획득, 실제 HTTP 연결을 요청

```
URL url = new URL(server + "/images/android-logo.jpg");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.connect();
```

HttpConnection 설정 메서드들

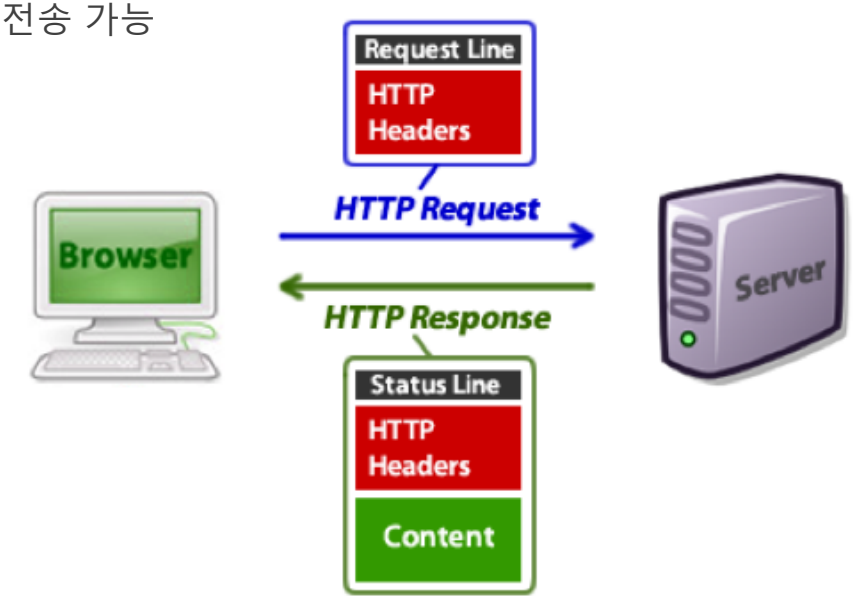
메서드	설명
setConnectTimeout	연결 타임아웃 시간 지정
setReadTimeout	읽기 타임아웃 시간 지정
setUseCaches	캐시 사용 여부 지정
setDoInput	데이터를 읽을 것인지 지정
setDoOutput	데이터를 요청 몸체에 쓸 것인지 지정
setRequestProperty	요청 헤더 지정
setRequestMethod	HTTP 요청 Method (GET, POST)

HTTP 통신

: GET과 POST, Request와 Response

- 서버에 데이터를 요청시 다음 두 가지 방식을 사용
  - GET 방식: 매개 변수를 URL에 붙여서 전송. 매개변수가 노출될 위험이 있고, 대량의 데이터, 텍스트 이외의 정보 등은 전송이 불가능
  - POST 방식 : 매개 변수를 Header에 담아서 전송. 매개변수가 쉽게 노출되지 않고, 대량의 데이터 혹은 바이너리(이미지, 파일 등) 데이터도 전송 가능

- 서버는 요청 내용에 따라 다음과 같은 내용을 반환
  - 상태값 : 처리 상황에 대한 코드값(200이 정상)
  - 헤더 : 반환 내용에 대한 정보
  - 콘텐츠 : 요청한 결과에 맞는 내용물



HTTP 통신
: 데이터의 송수신

- HttpURLConnection 접속이 이루어지면 connection으로부터 Stream을 얻어올 수 있고, java.io 패키지의 Stream 클래스들을 이용, 데이터를 송수신

```
InputStream is = conn.getInputStream();
bitmap = BitmapFactory.decodeStream(is);
is.close();
```

- 네트워크 통신은 많은 예외 상황이 있고, 해당 예외들을 잘 처리해 주어야 함
  - MalformedURLException
  - IOException 등

HTTP 통신

[실습] ImageViewExampleNetwork

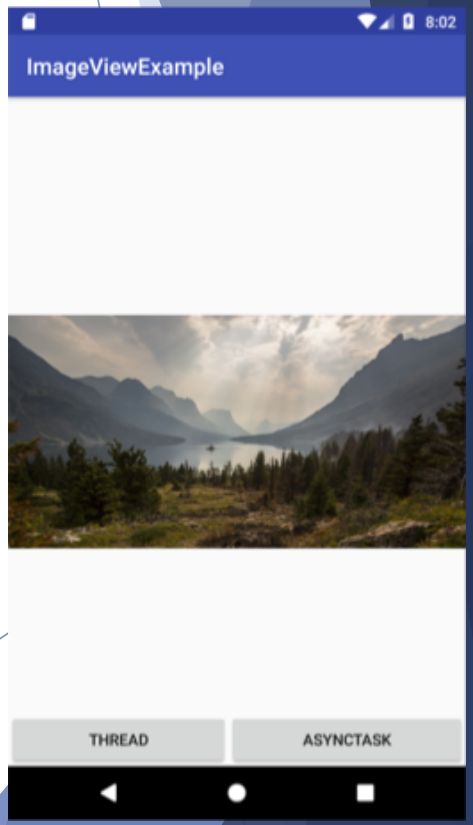
1) 프로젝트를 만들고 activity_main.xml 내용을 교체한 후, 레이아웃을 확인합니다.

2) THREAD 버튼 (btnThread)의 클릭 이벤트 (onBtnThreadClick) 내에 Thread를 이용,
SimpleAPIServer 서버 내의 다음 이미지를 받아 imageView에 표시합니다.

- /images/android-logo.jpg

3) ASYNCTASK 버튼 (btnAsyncTask)의 클릭 이벤트 (onBtnAsyncTaskClick) 내에 AsyncTask를 이용,
SimpleAPIServer 서버 내의 다음 이미지를 받아 imageView에 표시합니다.

- /images/landscape.jpg



HttpURLConnection의 단점

- HTTP 통신을 이용한 서버 연동은 표준 라이브러리인 HttpURLConnection 클래스를 이용, 모두 구현 가능
  - 하지만 단말기별 예외 처리, 네트워크 상황별 예외 처리, 쓰레드 혹은 AsyncTask의 구현 등 복잡한 상황이 발생 -> 실제 로직보다 예외 처리에 더 많은 시간 투자
  - 실무에서는 http-request 등 별도의 서드파티 라이브러리들을 이용

Volley API

- 2013년 Google I/O 행사에서 공개한 API
  - 안드로이드 앱에서 HTTP 통신을 좀 더 쉽게 구현하는데 목적

- Volley API는 표준 API가 아니므로 그레이들 파일에 dependency 관계를 설정하고 사용해야 함

```
//  volley는 표준 API가 아니므로 Gradle에 등록 후 sync
implementation 'com.android.volley:volley:1.0.0'
```

https://developer.android.com/training/volley/index.html

Volley API

: 핵심 클래스

클래스	설명
`RequestQueue`	Request 클래스의 정보대로 서버에 요청을 보내는 큐
`StringRequest`	문자열을 결과로 받는 요청
`ImageRequest`	이미지를 결과로 받는 요청
`JsonObjectRequest`	JSONObject를 결과로 받는 요청
`JsonArrayRequest`	JSONArray(JSON 배열)을 결과로 받는 요청

<Volley Request를 RequestQueue에 담는 예제>

```
RequestQueue queue = Volley.newRequestQueue(this);
JsonObjectRequest request = new JsonObjectRequest(...);
queue.add(request);
```

Volley API

: 주요 Request

<StringRequest의 생성>

```
StringRequest request = new StringRequest(Request.Method.GET,
        url,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {

            }
        },
        new Response.ErrorListener() {

            @Override
            public void onErrorResponse(VolleyError error) {

            }
        });
```

Volley API

: 주요 Request

<JsonObjectRequest의 생성>

```
JsonObjectRequest request = new JsonObjectRequest(
        Request.Method.GET, //  Request Method
        url,    //  Server URL
        null,   //  Json Request
        new Response.Listener<JSONObject>() {   //  Response Handler
            @Override
            public void onResponse(JSONObject response) {

            }
        },
        new Response.ErrorListener() {  //  Error Handler
            @Override
            public void onErrorResponse(VolleyError error) {

            }
        });
```

Volley API

: 주요 Request

<ImageRequest의 생성>

```
ImageRequest request = new ImageRequest(imageUrl,    //  Server URL
        new Response.Listener<Bitmap>() {    //  Response Handler
            @Override
            public void onResponse(Bitmap response) {

            }
        },
        0,   //  maxWidth
        0,   //  maxHeight
        ImageView.ScaleType.CENTER_INSIDE,   //  Image ScaleType
        Bitmap.Config.RGB_565,   //  Bitmap Mode
        new Response.ErrorListener() {   //  Error Handler
            @Override
            public void onErrorResponse(VolleyError error) {

            }
        });
```

JSON

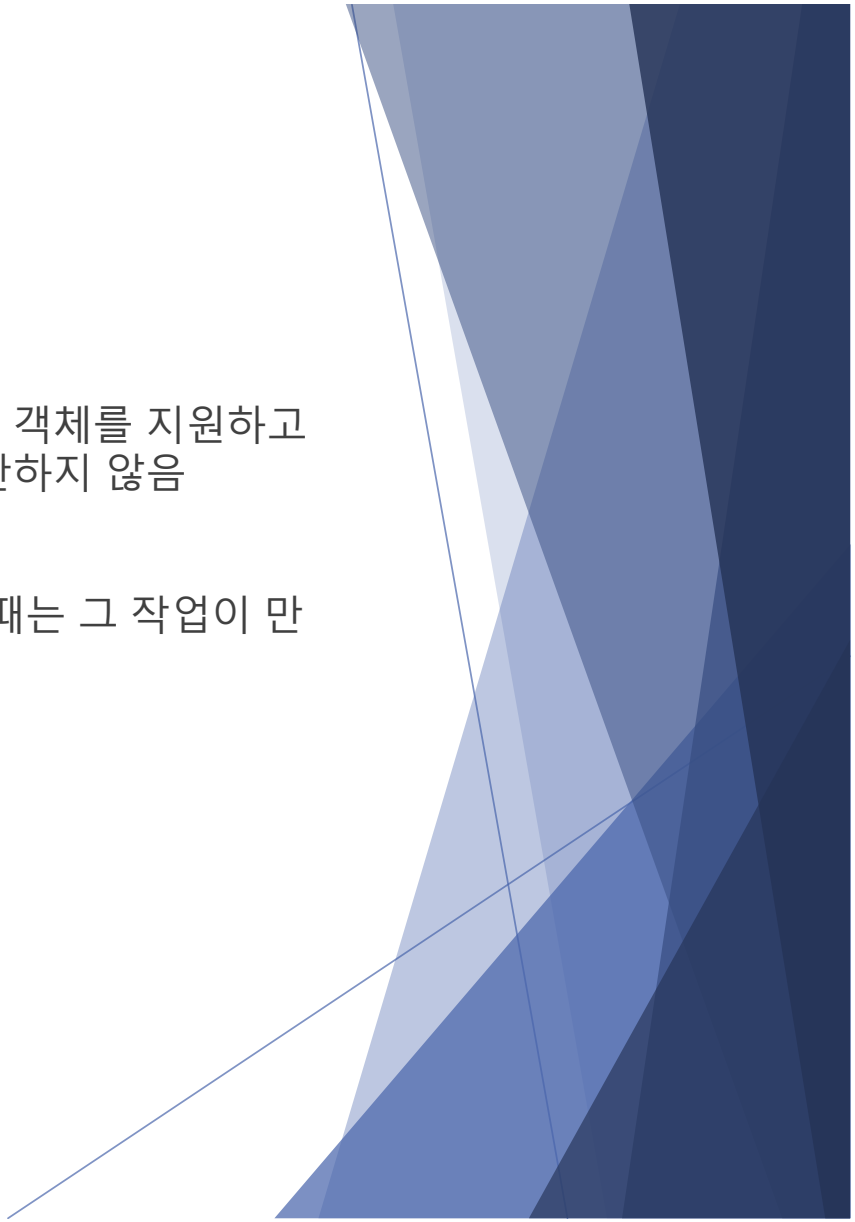
- JavaScript Object Notation
  - 최근에는 API 결과값을 전송할 때 XML보다 JSON을 더 많이 쓴다
  - 경량이고, 구조가 단순하여 쉽게 처리할 수 있다
  - 문자형, 수치형, 논리값, 배열, 사전 등 다양한 데이터형 지원


```
{
    hey: "guy",
    anumber: 243,
  - anobject: {
        whoa: "nuts",
      - anarray: [
            1,
            2,
            "thr<h1>ee"
        ],
        more: "stuff"
    },
    awesome: true,
    bogus: false,
    meaning: null,
    japanese: "明日がある。",
    link: http://jsonview.com,
    notLink: "http://jsonview.com is great"
}
```

JSON
: 그러나…

- JSON이 간단하고 Java가 JSONObject, JSONArray 등의 처리용 객체를 지원하고 있지만, 실제 JSON에서 데이터를 추출하여 사용하는 것은 간단하지 않음

- 특히 JSONArray 로 넘어온 데이터를 목록으로 저장하고자 할 때는 그 작업이 만만치 않음



JSON

: Google Gson의 도입

- Gson의 사용
  - 그레이들 파일에 dependency 를 설정한 후 사용해야 함

```
compile 'com.google.code.gson:gson:2.8.2'
```

- Gson은 JSONObject, JSONArray로부터 데이터 모델 클래스를 매칭, 해당되는 값들을 반영하여 새 객체를 만들어줌
  - 모델이 될 클래스에 필드를 선언하고 getter/setter 설정

```
class Movie {
    private Integer id;
    private String title;
    private String director;
    private String image;
    private Integer year;

    public Integer getYear() {
        return year;
    }

    //  ... getters and setters
```

JSON

: Google Gson - JSON을 객체로 연결하기

- Gson 객체를 생성하고 fromJson 메서드를 이용, 클래스와 연결하여 실제 데이터 객체를 만들어냄

```
Gson gson = new Gson();
String json = "{\"id\":1,\"title\":\"토이 스토리\",\"director\":\"존
라세터\",\"year\":1995,\"image\":\"toystory.jpeg\"}";
Movie movie = gson.fromJson(json, Movie.class);
```

- toJson 메서드를 이용하면 데이터 객체로부터 JSON 데이터를 만들어낼 수도 있음

- JSONArray 객체의 parsing 예

```
Gson gson = new Gson();
Movie[] array = gson.fromJson(movies.toString(), Movie[].class);
```

REST API와 JSON을 이용한 네트워크 앱

[실습] MovieListApp

1) 프로젝트를 만들고 activity_main.xml 내용을 교체한 후, 레이아웃을 확인합니다.

- activity_main.xml : 메인 액티비티 레이아웃
- list_item_movie.xml : 리스트에 표시할 아이템의 레이아웃

2) SimpleAPIServer의 movies JSON 형식을 확인하고 다음과 같은 모델 클래스를 생성합니다.
GSon에서 사용할 클래스이므로 단순 VO가 아닌 getter/setter를 가진 클래스입니다

- 클래스명 : Movie
- 필드 목록

```
private Integer id;
private String title;
private String director;
private String image;
private Integer year;
```

REST API와 JSON을 이용한 네트워크 앱

[실습] MovieListApp

3) ListView의 콘텐트 뷰를 담아둘 Holder 클래스를 생성합니다.

- 클래스명 : MovieHolder
- list_item_movie 레이아웃과 연결시킬 위젯 변수들을 담고 있습니다.

4) Volley API의 JsonObjectRequest를 생성하고 SimpleAPIServer의 다음 API를 구현합니다.

- API : /movies

5) JsonObjectRequest의 Response.Listener 내에서 movies JSONArray를 Gson으로 파싱, ArrayList로 만듭니다.

6) Volley API의 RequestQueue에 요청을 의뢰, JSON 데이터가 원활히 ArrayList로 변환되는지 확인

REST API와 JSON을 이용한 네트워크 앱

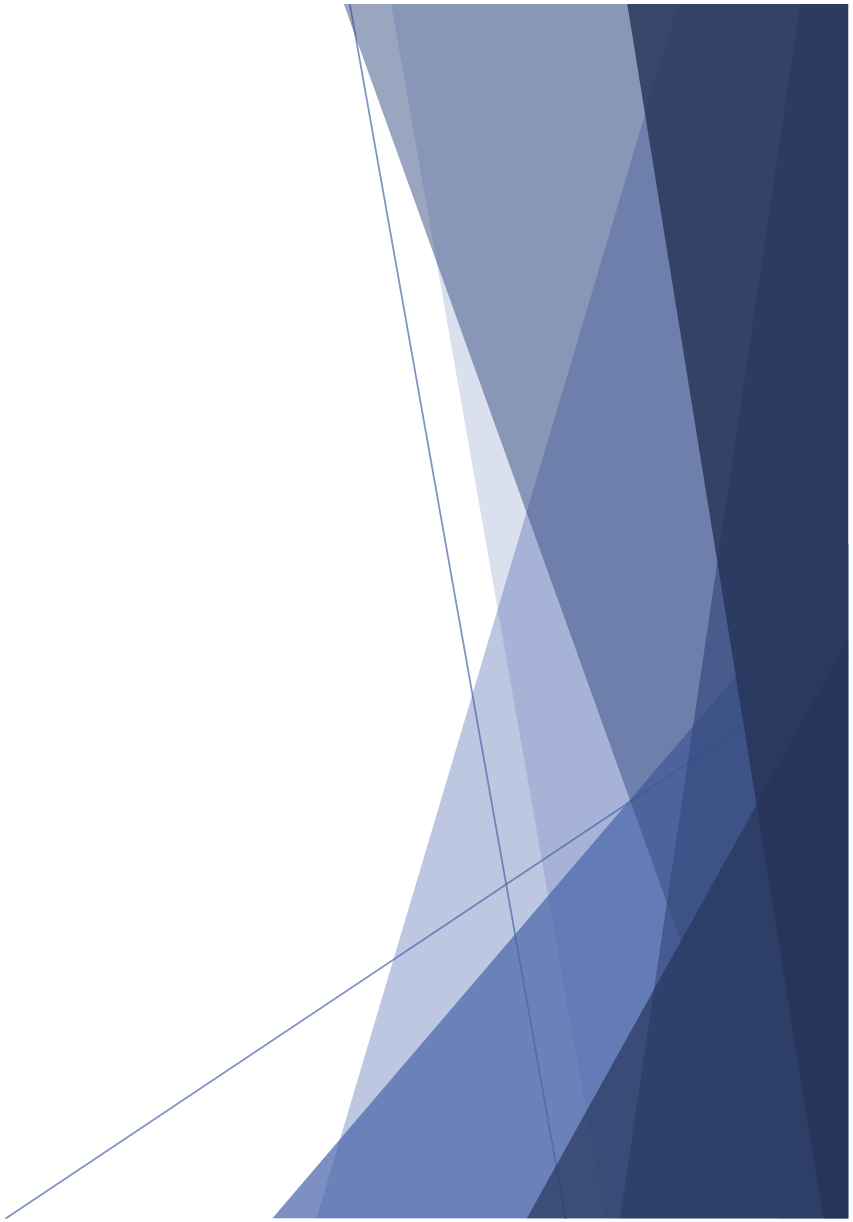
[실습] MovieListApp

7) ArrayAdapter를 상속받은 MovieAdapter를 구현하고 앞에서 만든 ArrayList를 연결합니다.

8) MovieAdapter를 MainActivity의 ListView에 연결합니다.

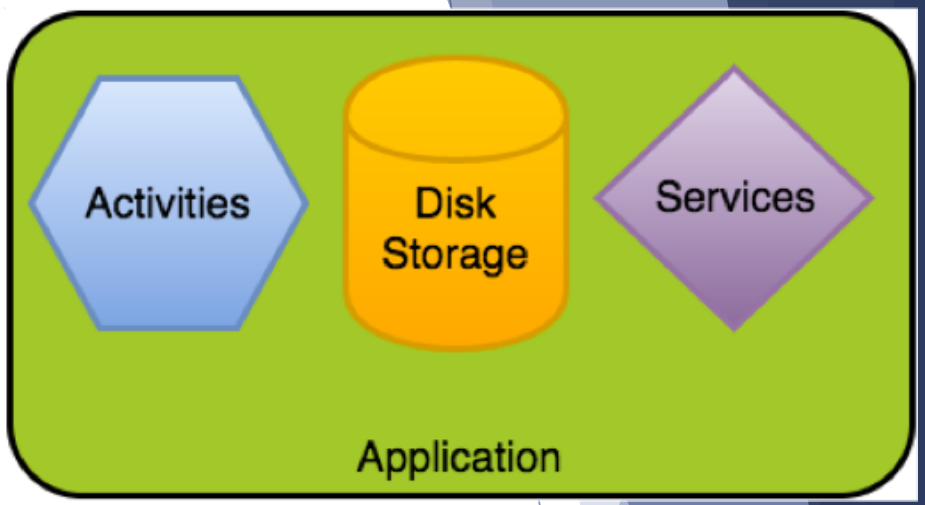
Manager의 구현

Singleton Pattern



Manager

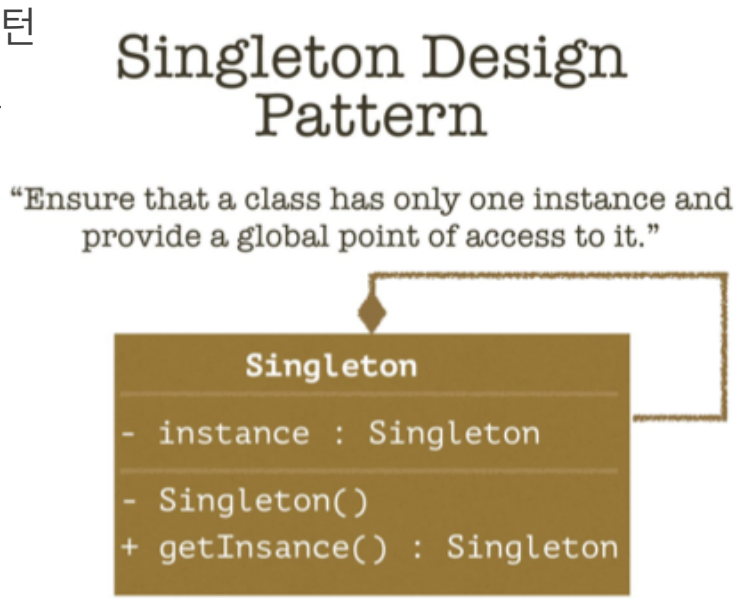
: Application 클래스와 Singleton



- Activity, Fragment 들은 각자의 메모리 공간을 점유
  - 각자의 필드와 메서드, 로직 들을 각자 가지고 있음
  - 그러나 실제 앱을 작성할 때에는 데이터를 공유하거나 스택, 큐 등을 함께 사용해야 할 필요
- 안드로이드는 앱 컴포넌트 사이에서 공유될 수 있는 Application 객체를 제공
  - Application 공간을 이용하려면 Application을 상속받은 클래스를 만들어 등록, 사용해야 함
- Application 클래스와 Singleton 패턴을 이용한 Manager를 만들어 데이터, 스택, 큐 등 공용 리소스들을 관리해 봅시다

Manager
: Singleton

- 전체 프로그램 상에서 동일한 인스턴스를 사용해야 할 경우 활용
  - 어떠한 상황에서도 **단 하나**의 인스턴스만 유지
- 4대 디자인 패턴에 포함될 정도로 유명하고, 유용한 패턴
  - 그 자체만으로도 유용하지만, 다른 패턴들을 구현할 때도 응용되어 많이 사용됨



Manager
: Singleton

- Singleton 클래스의 작성
  - 현재 클래스의 인스턴스를 저장할 정적 멤버를 선언
  - 생성자를 private으로 하거나 static으로 선언
    - 인스턴스에서 직접 new 할 수 없도록 제한
    - 인스턴스를 받아오려면 getInstance() 메서드를 이용

```
class MovieManager {
    private static final MovieManager ourInstance = new MovieManager();

    static MovieManager getInstance() {
        return ourInstance;
    }

    private MovieManager() {
    }
}
```

Manager
: Application

- Application 상속 클래스의 작성
  - 공유 영역 사용을 위해 Application을 상속받은 MyApplication 클래스를 작성

```
public class MyApplication extends Application {
    private static Context context;
    @Override
    public void onCreate() {
        super.onCreate();
        this.context = this;
    }

    public static Context getContext() {
        return context;
    }
}
```

  - 작성한 MyApplication 클래스는 Manifest의 application 에 클래스를 등록 후 사용

```
<application
    android:name=".MyApplication"
```

Manager

: Application

[실습] MovieListApp

1) 현재 MovieListApp은 Volley API의 RequestQueue를 매 순간 중복 선언 사용중입니다.
이 부분을 개선해 봅니다.

2) Application을 상속받은 MyApplication 클래스를 구현, Application의 공용 Context를 이용할 수 있도록 수정해 봅니다.

3) Singleton 패턴을 응용한 MovieManager 클래스를 만들고, Volley API의 RequestQueue를 앱 전체 영역에서 공유할 수 있도록 합니다.

4) MovieListApp 내에서 Volley API의 RequestQueue를 생성하고 add 하는 부분을 모두 MovieManager의 Queue에서 수행할 수 있도록 변경해 봅니다.