

SHOUT · OUR · PASSION · TOGETHER

ODo IT SOPT O

# 2 차 세 미 나

SHOUT OUR PASSION TOGETHER  
**SOPT**

# 01

## 안드로이드 4대 컴포넌트와 Intent

# 02

## Activity 실습

# 03

## Fragment 알아보기, 실습

# 04

## 더 알려드려요~!

- `FragmentStatePagerAdapter`
- 싱글톤
- `Bundle()`



01



# 안드로이드

## 4대 컴포넌트와 Intent

## 4대 컴포넌트와 Intent

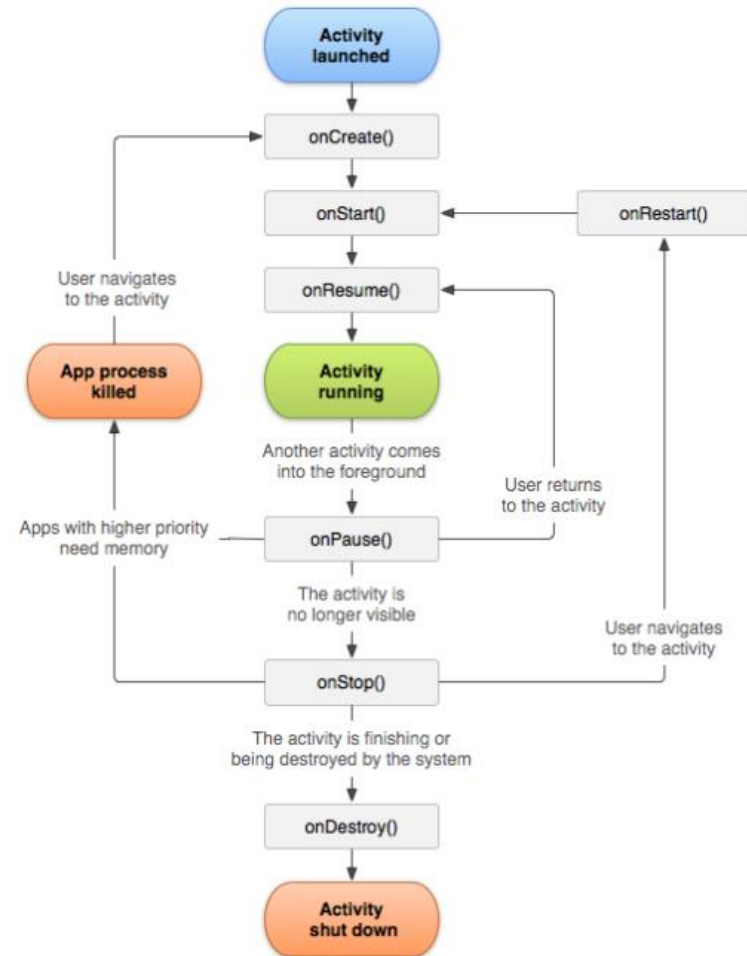
### 안드로이드 컴포넌트

- Activity
- Service
- Content Provider
- Broad Cast Receiver

### Intent

- 일종의 메시지 객체
- 안드로이드 컴포넌트 간 통신 매체

- UI가 있는 앱 단일 화면
- 하나의 앱에 여러 Activity가 존재하지만 각자 독립적으로 존재
- 생명주기를 가짐
- manifest파일에 <activity>를 통해 선언

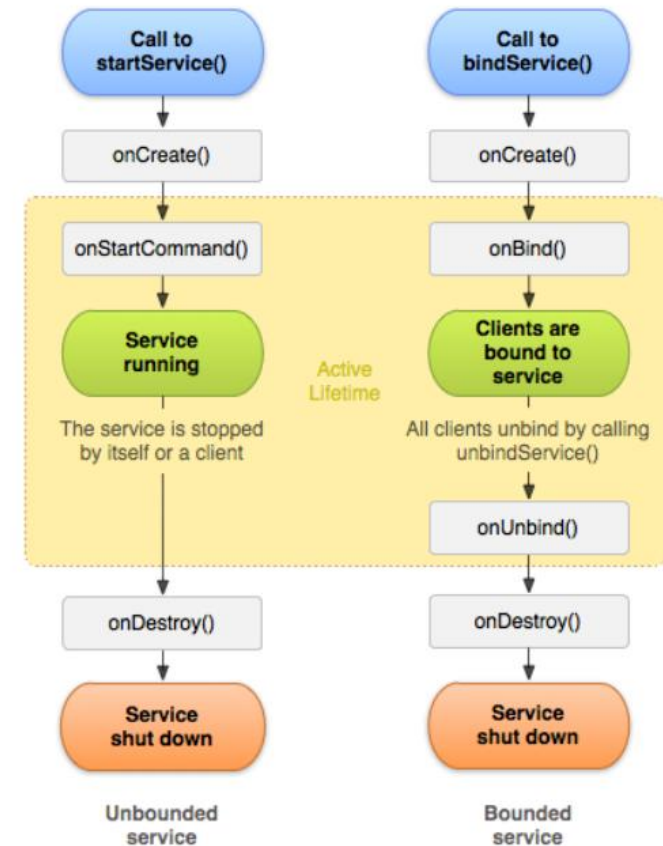


**Figure 1.** A simplified illustration of the activity lifecycle.

## Activity 생명주기

# Service

- UI가 제공되지 않음
- 백그라운드에서 실행되는 구성 요소
- 오랫동안 실행되는 작업이나 원격 프로세스를 위한 작업 수행
- 생명주기를 가짐
- 대표적인 Service로는 백그라운드에서 음악 재생
- manifest파일에 <service>를 통해 선언



Service 생명주기

# Content Provider

- **공유된** App 데이터 집합 관리
- 우리의 여러 다른 앱끼리 데이터를 공유
- 공유자가 허용한다면 타 앱에서 데이터를 쿼리하거나 수정까지 가능
- 대표적으로 안드로이드 시스템은 사용자의 연락처 정보를 관리하는 Content Provider를 제공, 다른 앱에서 연락처 정보를 읽기 가능

# Broad Cast Receiver

- 시스템 범위의 알림에 응답하는 구성 요소
- 예를 들어, 화면 꺼질 때와 켜질 때, 배터리 잔량 부족, 화면 캡처 등에 대한 알림을 받고 응답할 수 있다.



# Intent

- 일종의 **메시지 객체!**
  - 안드로이드 구성 요소에 작업을 요청할 수 있음
  - 명시적 인텐트와 암시적 인텐트, 두 가지 인텐트가 존재
  - **액션**과 **데이터**, 카테고리 구성되어 있다.
- 
- 명시적 인텐트: App내 특정 Activity나 Service 등 특정 구성 요소를 시작하기 위해 사용
  - 암시적 인텐트: 기기 내에서 인텐트에 명시한 작업을 수행할 수 있는 모든 App을 호출 가능

○

02

○

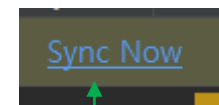
# Activity 실습

## 02 Activity 실습

### 쉬는 시간 동안 실습 프로젝트 Setting하기!!!

- (선택1-1) 프로젝트 생성 후 compile, target SDK Version 26으로 바꿔보세요!

```
android {  
    compileSdkVersion 26  
    defaultConfig {  
        applicationId "com.sopt_nyh.seminar_002_example"  
        minSdkVersion 26  
        targetSdkVersion 26  
        versionCode 1  
        versionName "1.0"  
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"  
    }  
    buildTypes {  
        release {  
            minifyEnabled false  
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'  
        }  
    }  
}
```



- (선택1-2) 그러면! 아래 라이브러리가 빨간 밑줄이 그어질 텐데!! 버전을 27.1.1에서 26.0.1로 바꾼 뒤 Sync

```
dependencies {  
    implementation fileTree(include: ['*.jar'], dir: 'libs')  
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"  
    implementation 'com.android.support:appcompat-v7:26.0.1'  
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'  
    testImplementation 'junit:junit:4.12'
```

- (필수) 아래 두 라이브러리를 추가해주세요!!!

```
implementation 'org.jetbrains.anko:anko:0.10.5'  
implementation 'com.android.support:design:26.0.1'
```

### <실습 1-1> Activity 2개 만들기

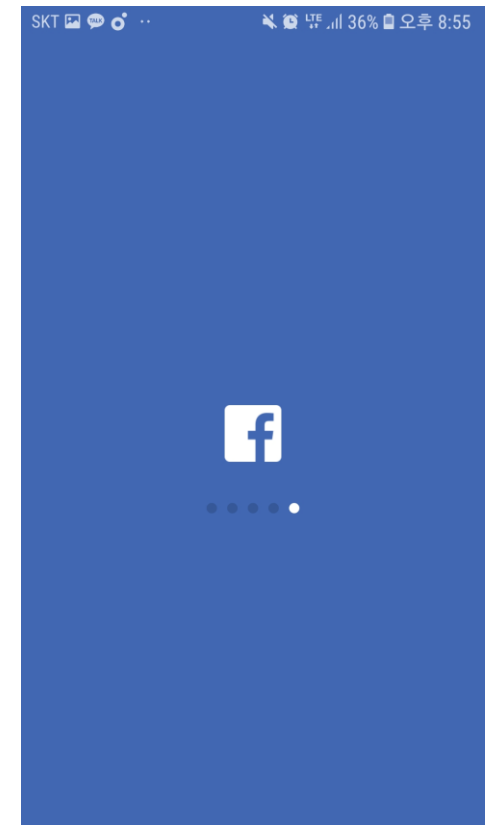
1. Splash Activity와 Sub Activity를 만드세요!
2. Splash Activity를 manifest파일을 통해 처음 띄워질 Activity로 변경!
3. (선택) 구글링을 통해 본인이 띄우고 싶은 이미지를 다운받아 뒤 배경 구성하기

#### tip. Splash 화면은 왜 쓸까?!

대표적인 이유로!

사용자에게 보여질 UI(Main Activity)에 대한 데이터들을 서버에서 받아올 때,  
딜레이가 생기겠죠???

그때! 준비 시간 동안 App 퍼포먼스를 위해…?



<페이스북 Splash 화면>

### <실습 1-2> Splash Activity에서 Main Activity 시작 시키기

- **startActivity():**

**intent** 메시지 객체를 통해 새로운 Activity를 시작 시킨다.

예제) Splash Activity에서 Main Activity 시작 시키는 코드

```
val intent : Intent = Intent(this@SplashActivity, MainActivity::class.java)
intent.putExtra("message", "헬로우 메인!")
intent.putExtra("myNumber", 100)
startActivity(intent)
```

“message”라는 **key**에  
“헬로우 메인!” 라는 **value**가 담겨 있다.

이처럼, Intent는 value를 구분할 수 있도록  
(key,value) 한 쌍으로 데이터를 이룬다.

```
startActivity<MainActivity>("message" to "헬로우 메인!", "myNumber" to 100)
```

동일한 코드.  
위는 오리지날  
아래는 anko 라이브러리 사용

- **startActivityForResult():**

**intent** 메시지 객체를 통해 새로운 Activity를 시작 시키고, 그 Activity가 끝날 때 결과를 얻고 싶을 경우 사용.

그 결과는 onActivityResult() 함수를 override하여 얻는다.

```
val intent : Intent = Intent(this, SubActivity::class.java)
intent.putExtra("message", "헬로우 서브!!")
startActivityForResult(intent, 7777)
```

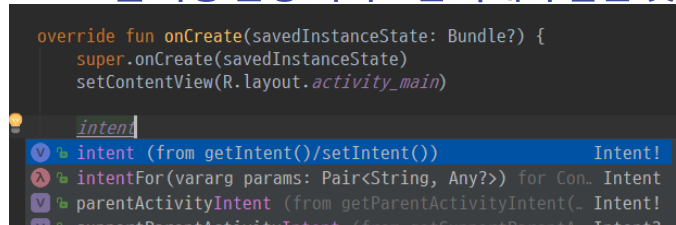
```
startActivityForResult<SubActivity>(7777, "message" to "헬로우 서브!!")
```

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

추후 이용 법을 알아보시다!!!

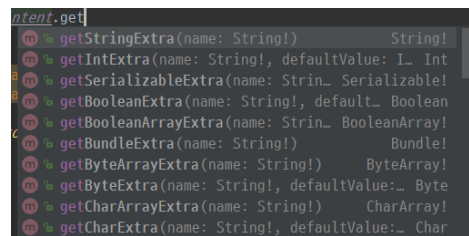
## <실습 1-3> 시작된 Main Activity에서 데이터 꺼내기

- 시작된 Main Activity에서 intent를 자동 완성 시켜보면 아래와 같은 것이 있을 것 입니다!



<Java와는 다르게 Kotlin에서는 getter, setter 함수를 사용하지 않고 바로 접근이 가능해요>

- intent 객체 내의 다양한 getExtra 메소드를 통해 우리 의도 대로 선택하여 데이터를 꺼내 먹어요!



- 아래 코드는 Main Activity의 onCreate 메소드에서 데이터를 꺼내어 toast로 띄워주는 예제입니다.

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
  
    var message : String = intent.getStringExtra("message")  
    var myNum : Int = intent.getIntExtra("myNumber", 0)  
  
    toast("$message, $myNum")  
}
```

but! 이대로 쓰지 말고  
null처리를 해봅시다!

### 그림으로 정리

1. intent에 데이터를 넣고  
**startActivity()**로 Main 시작
2. 그리고 **finish()**로 Splash 종료

getExtra()로 데이터 받음




### <실습 1-4> **startActivityResult()** 알아보자! Main Activity에서 Sub Activity 시작 시키기!

- Main Activity에서 startActivityResult()로 Sub Activity 띄우기!

```
val intent : Intent = Intent(this, SubActivity::class.java)
intent.putExtra("message", "헬로우 서버!!")
startActivityResult(intent, REQUEST_CODE_SUB_ACTIVITY)
```

```
startActivityResult<SubActivity>((REQUEST_CODE_SUB_ACTIVITY, "message" to "헬로우 서버!!"))
```

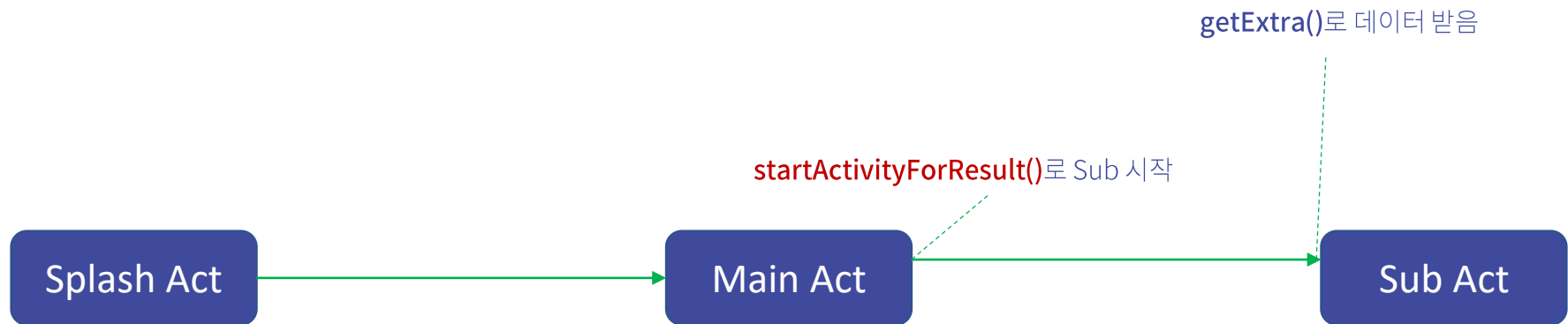


```
val REQUEST_CODE_SUB_ACTIVITY = 7777
```

나중에 어떤 Activity에서 보내는 결과인지 알아보기 위해서 구별하기 위해 사용되는 Request Code 입니다!!!  
보통 클래스 내 인스턴스 상수로 만들어서 사용해요~! 상수 명과 값은 본인 마음대로~  
but 알아보기 쉽게 이름 짓기



### 그림으로 정리



### <실습 1-5> Sub Activity에서 Main Activity로 결과 보내기!

- Sub Activity에서 intent에 값을 넣고 보낸 뒤 **종료(finish())** 시켜 봅시다!

```
btn_sub_act_finish.setOnClickListener {  
    val intent : Intent = Intent()  
    intent.putExtra("result", "SubAct가 주는 데이터!")  
    setResult(Activity.RESULT_OK, intent)  
    finish()  
}
```

옆 예제에서는,  
Sub Activity를 종료 시키는 기능을 하는  
Button 하나를 만들었습니다!

Activity.RESULT\_OK는 결과를 전달 시킬 타겟(Main Act)에 보내는 Result Code인데,  
Main Activity가 이 Result Code을 받고, 값에 따라서 로직을 짤 수 있어요~!  
사용 예는 다음 장에서!

여기서 주요하게 볼 점은 **setResult()** 메소드를 통해 intent에 데이터를 넣어  
Main Activity로 결과 값을 전달한다는 점!!!

### <실습 1-6> Sub Activity에서 보낸 결과 데이터, Main Activity에서 onActivityResult()를 통해 받아보자!

- Main Activity에서 onActivityResult를 override하여 로직을 짭니다!

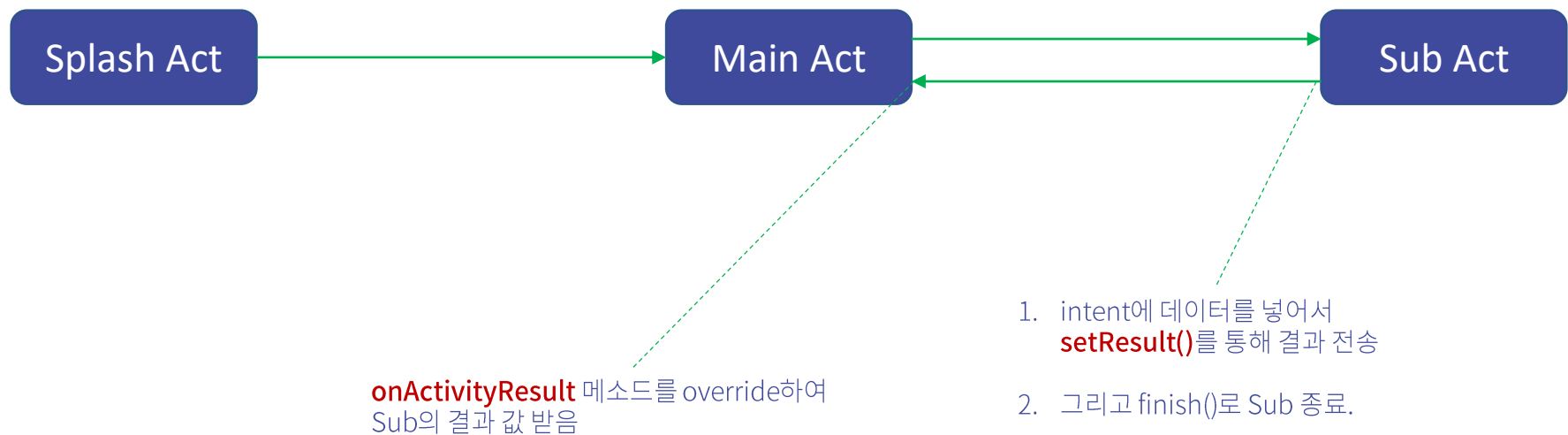
```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
    super.onActivityResult(requestCode, resultCode, data)  
  
    if ((requestCode == REQUEST_CODE_SUB_ACTIVITY)){  
  
        if (resultCode == Activity.RESULT_OK){  
            tv_main_act_text.text = data!!.getStringExtra("result")  
        } else {  
            tv_main_act_text.text = "제대로 값 못받았어!!! RESULT_OK가 아니래!"  
        }  
    }  
}
```

일단!  
보내진 결과가 Sub Activity에서  
보내진 것인지 requestCode로 판단하고!

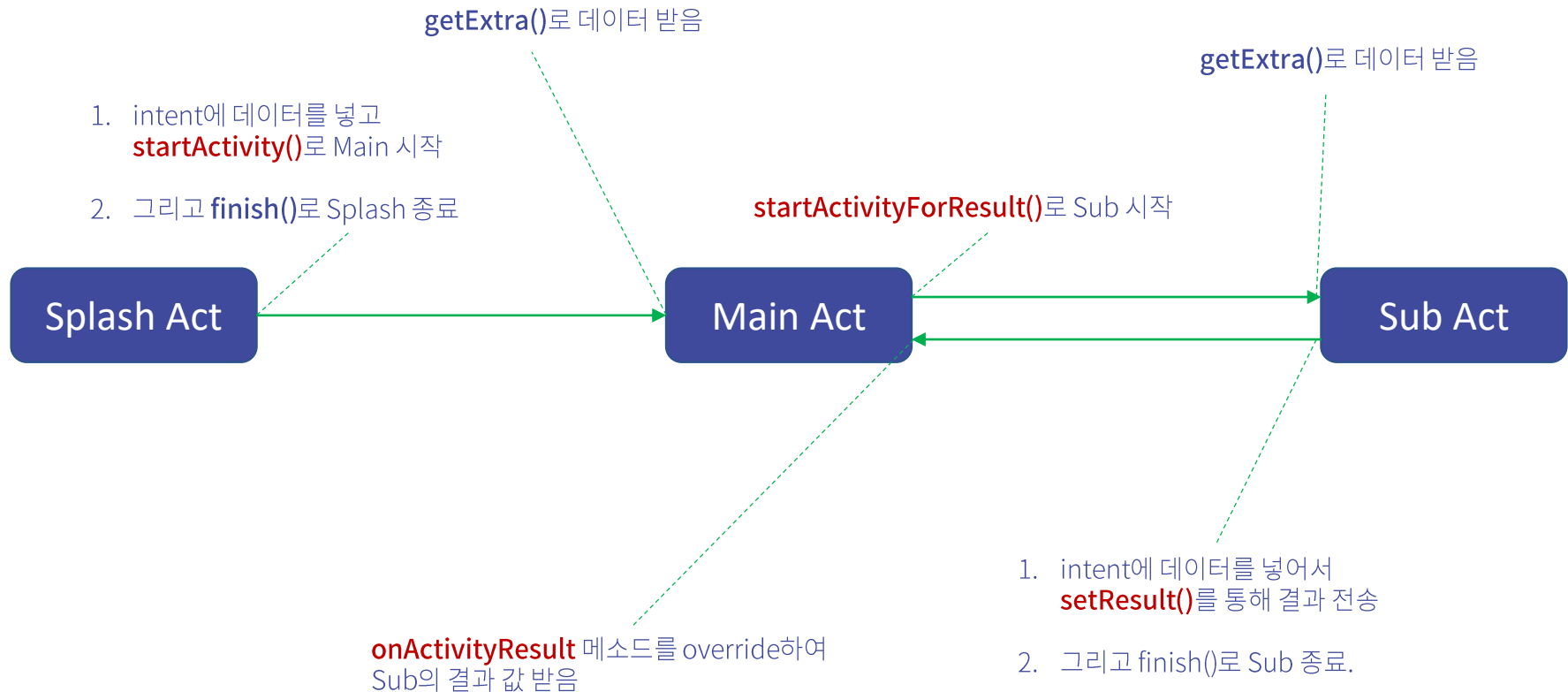
Sub Activity가 어떤 resultCode를  
보냈는지 판별한 후

RESULT\_OK일 때  
처리할 로직 수행!  
여기서는 intent로 보낸 key가 result인 데이터를  
꺼내서 textView에 넣어줬어요!

### 지금까지 한 작업 그림으로 정리



### 지금까지 한 작업 그림으로 총 정리



### (+추가 코드) back 버튼 처리

- Main Activity에 두 번 back 버튼을 눌렀을 때만 종료되도록 바꿔주는 코드를 넣어보자!

1. Main Activity의 인스턴스 변수로 backPressedTime 변수를 추가해보자!

```
var backPressedTime: Long = 0
```

2. onBackPressed 메소드를 override하여 2초 내 두 번 눌렀을 때만 종료되도록 바꿔보기

```
override fun onBackPressed() {  
    var temp: Long = System.currentTimeMillis()  
    var intervalTime: Long = temp - backPressedTime  
  
    if (intervalTime in 0..2000) {  
        super.onBackPressed()  
    } else {  
        backPressedTime = temp  
        toast("버튼을 한번 더 누르면 종료됩니다.")  
    }  
}
```



03



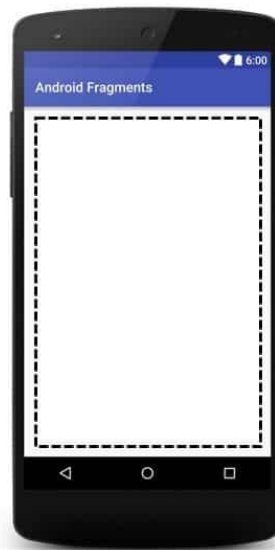
# F r a g m e n t

### Fragment란?

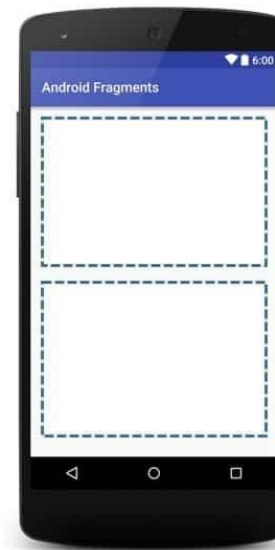
- 다른 액티비티에서 재사용할 수 있는 “하위 액티비티”와 같은 개념  
즉, 액티비티 내 UI를 그룹화 하여 떼어낸 느낌...
- 액티비티가 스케치북 종이 한 장이라면, Fragment는 그 종이 한 장 위에 올려지는 **조각 스케치북**  
(그 조각의 크기는 최대 액티비티와 같을 수 있음)



Activity without Fragment



Fragment inside an Activity

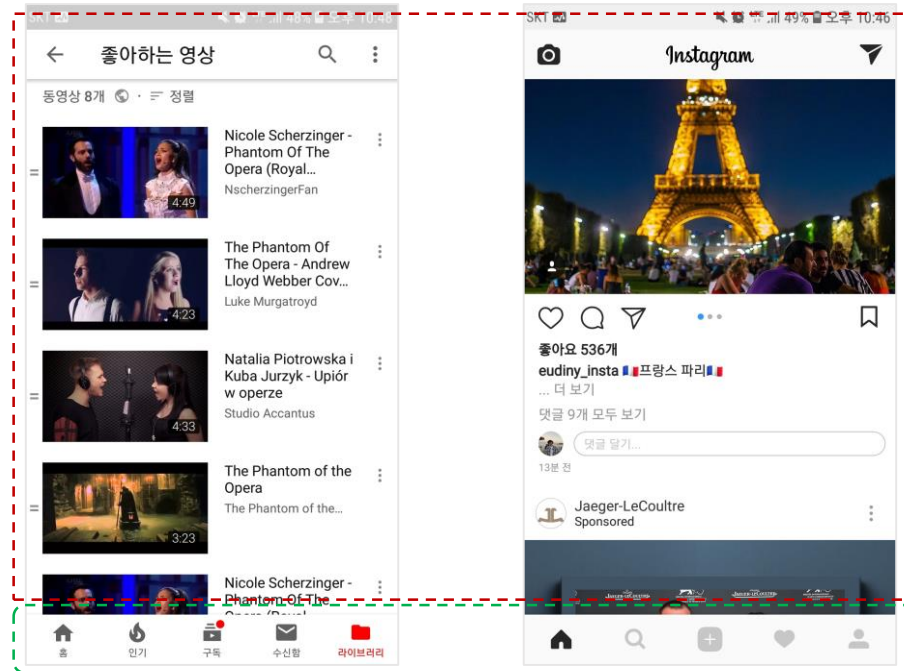


Activity holding two  
Fragments



## Fragment 왜 쓸까?

1. 앱에서 공통적으로 쓰이는 UI를 Fragment로 떼어 다른 Activity에서 재사용한다.
2. 하나의 Activity위에 다양한 UI 전환을 위해, 여러 Fragment를 이용한다.  
(2번의 예를 들면, 아래의 사진과 같다.)

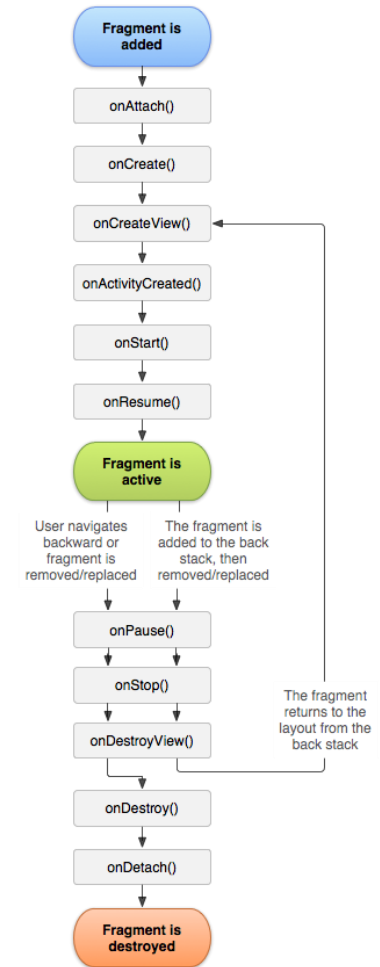


Fragment로 구성

탭을 누를 때,  
Activity 교체 없이  
빨간 박스 UI만 변화한다.

### Fragment 더 알아보기

1. 앞서 하위 액티비티 느낌이라고 말했 듯, **생명주기**가 존재한다.
2. Fragment는 호스트 액티비티의 생명주기에 직접적으로 영향 받는다.  
(만약 Main Activity 위에 올려진 Fragment라면 Main Activity가 소멸되면 그 위 모든 Fragment도 소멸)
3. Fragment 위에 Fragment를 올릴 수 있다.  
(너무 얹히고 얹히면 고통스러우니까 적당히 쌓아 올리자..!)
4. Activity는 Intent로 데이터를 전달하지만, Fragment는 **Bundle**이란 객체 사용.

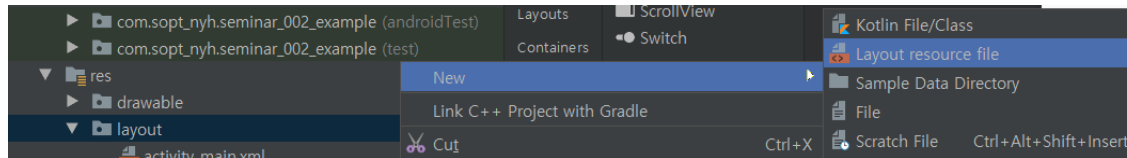


Fragment 생명주기

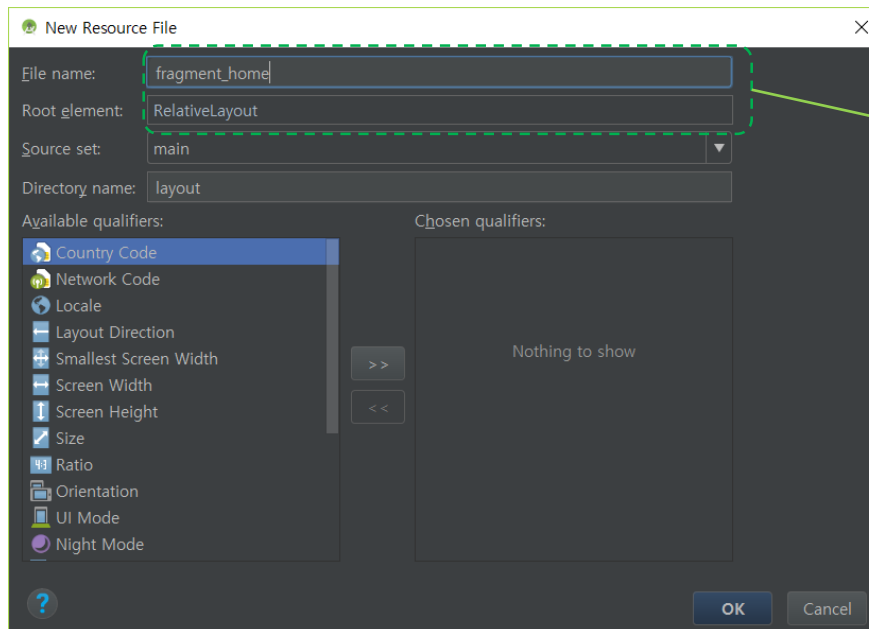
# Fragment 실습

## 첫번째, UI 조각 만들기 -Fragment Layout 만들자!

1. res → layout → 마우스 오른쪽 → New → Layout resource file



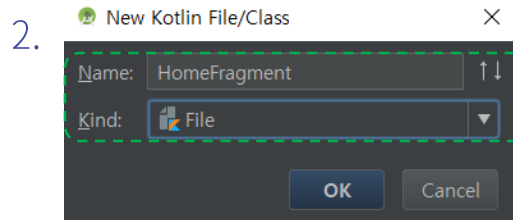
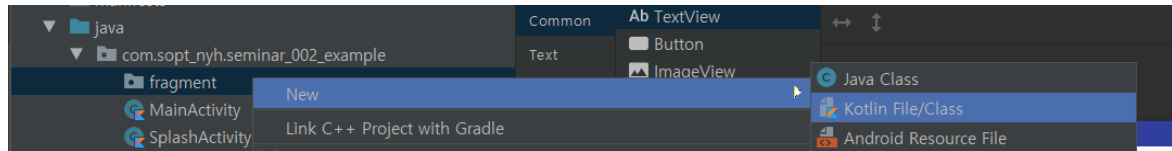
- 2.



1. 파일 이름은 “fragment\_home”  
(대분자, -, 띄어쓰기 안됩니다!!!!)
2. root layout은 RelativeLayout  
(본인이 편한 것으로 해도 돼요!)

### 두번째, Fragment class 만들기 - 1

1. fragment 디렉토리 하나 만든 뒤 → new → Kotlin File/Class



1. 파일 이름은 “HomeFragment”
2. 종류는 아무거나 해도 나중에 코드에 따라 안드로이드 스튜디오가 바꿔줍니다! 똑똑하죠?

### 두번째, Fragment class 만들기 - 2 (Layout 붙여 주기)

1. 되도록 자동완성으로 코딩하도록 해요!!! fragment\_home.xml 붙이기 끝!

```
class HomeFragment : Fragment() {  
    override fun onCreateView(inflater: LayoutInflater?, container: ViewGroup?, savedInstanceState: Bundle?): View? {  
        val homeFragmentView : View = inflater!!.inflate(R.layout.fragment_home, container, false)  
        return homeFragmentView  
    }  
}
```

2. 자동완성이 잘 안되거나 에러 뜨면 아래 import 참조해보세요!

```
import android.os.Bundle  
import android.support.v4.app.Fragment  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import com.sopt_nyh.seminar_002_example.R
```

```
class HomeFragment : Frag|  
    Fragment (android.app)  
    Fragment (android.support.v4.app)  
    FragmentManager (android.app)
```

### 세번째, Main Activity에 Home Fragment 붙이기!

1. activity\_main.xml에 Fragment가 들어갈 틀 만들어 주기!

```
<FrameLayout
    android:id="@+id/fl_main_act_fragment_block"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</FrameLayout>
```

2. MainActivity.kt에 Fragment를 붙이기 위한 코드 작성(재사용을 위해 함수로)

```
private fun addFragment(fragment : Fragment){
    val transaction : FragmentTransaction = supportFragmentManager.beginTransaction()
    transaction.add(R.id.fl_main_act_fragment_block, fragment)
    transaction.commit()
}
```

3. onCreate에서 해당 함수에 HomeFragment() 인스턴스를 넣고 호출해보자!

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    addFragment(HomeFragment())
}
```

### 코드 해석

```
private fun addFragment(fragment : Fragment){  
    val transaction : FragmentTransaction = supportFragmentManager.beginTransaction()  
    transaction.add(R.id.fl_main_act_fragment_block, fragment)  
    transaction.commit()  
}
```

1. **액티비티 내** Fragment는 FragmentManager에 의해 관리됩니다. (add(), remove(), hide(), show(), replace())  
(UI 재사용을 위해 Fragment를 사용할 땐 위에 설명한 함수들을 이용해 관리하세요!)
2. 프래그먼트 내 Fragment는 ChildFragmentManager로 관리합니다.
3. 트랜잭션을 통해 관리가 이루어 집니다. commit을 통해 완료를 알리죠!



### 네번째, 이제는 Fragment의 UI를 건드려봅시다!

1. fragment\_home.xml에 테스트로 쓸 RelativeLayout과 TextView를 하나씩 추가해 봅시다! id값 주세요!

```
<RelativeLayout
    android:id="@+id/rl_main_act_home_frag_background"
    android:layout_margin="16dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</RelativeLayout>
<TextView
    android:id="@+id/tv_main_act_home_frag_title"
    android:text="Home Fragment 입니다."
    android:layout_centerInParent="true"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

2. 프래그먼트 생명주기를 잘 참조해보면 View가 생성 된 후 View를 건들 수 있는 곳은  
onActivityCreated 메소드 임을 알 수 있습니다. 그러므로 해당 생성주기 단계에서 View를 건드려보죠!

```
override fun onActivityCreated(savedInstanceState: Bundle?) {
    super.onActivityCreated(savedInstanceState)

    rl_main_act_home_frag_background.backgroundColor = Color.parseColor("#BBDEFB")
    tv_main_act_home_frag_title.textSize = 18f
}
```

### 실습 더 해보자!

1. UserFragment 만들기(fragment\_user.xml도 만들어야 겠죠? 내용물은 자유롭게!)

2. MainActivity에 Fragment를 replace시키는 함수 만들기!

```
private fun replaceFragment(fragment: Fragment){  
    val transaction : FragmentTransaction = supportFragmentManager.beginTransaction()  
    transaction.replace(R.id.fl_main_act_fragment_block, fragment)  
    transaction.commit()  
}
```

3. MainActivity에 버튼 2개(HomeButton, UserButton) 만들고 클릭 리스너를 만든 뒤, replaceFragment 함수를 통해 Fragment 교체 시켜 보자!

## 03 Fragment 실습

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    addFragment(HomeFragment())

    setViewClickListener()
}

private fun setViewClickListener() {
    btn_main_act_home_btn.setOnClickListener {
        replaceFragment(HomeFragment())
    }

    btn_main_act_user_btn.setOnClickListener {
        replaceFragment(UserFragment())
    }
}

private fun addFragment(fragment: Fragment) {
    val transaction: FragmentTransaction = supportFragmentManager.beginTransaction()
    transaction.add(R.id.fl_main_act_fragment_block, fragment)
    transaction.commit()
}

private fun replaceFragment(fragment: Fragment) {
    val transaction: FragmentTransaction = supportFragmentManager.beginTransaction()
    transaction.replace(R.id.fl_main_act_fragment_block, fragment)
    transaction.commit()
}
```



04

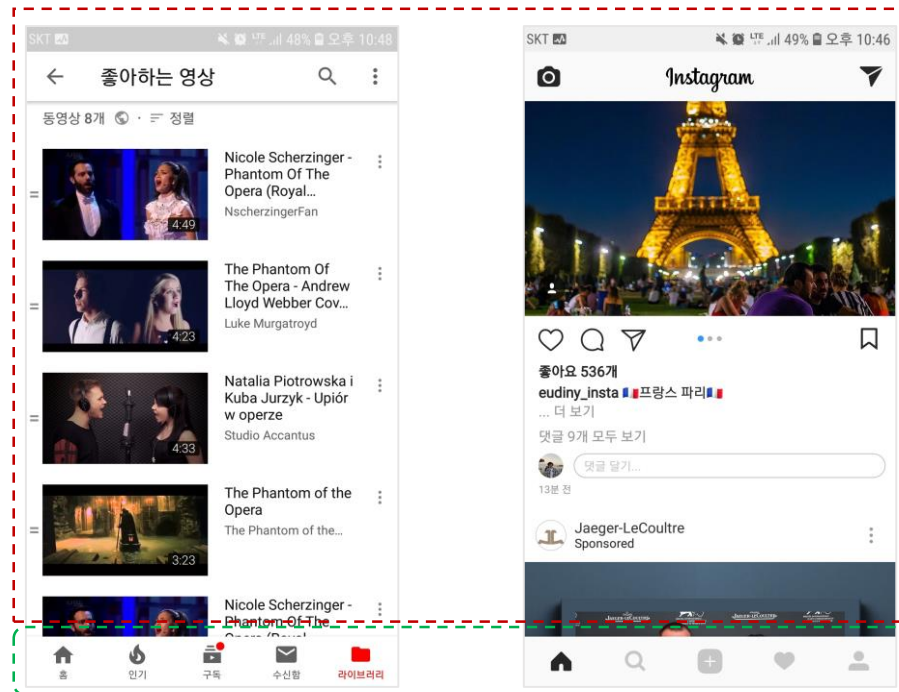


더 알려드려요 ~ !

### FragmentStatePagerAdapter란?

1. 일단, 안드로이드에서 Pager라는 단어가 View에 들어가면 책장을 넘기듯 화면을 slide 시키는 View라고 생각하면 됩니다!
2. FragmentStatePagerAdapter는 slide되는 view가 Fragment인 pager에서 Fragment를 관리해주는 Adapter입니다!!! 말이 어렵죠?  
➔ 그냥 우리는 Fragment가 슬라이드 되는 View를 만들 것인데 그것을 관리해주는 그런 겁니다!!!!...
3. FragmentPagerAdapter란 것도 있는데, 이것은 고정된 개수의 Fragment에 적합한 Adapter입니다!  
한번 생성되면 Fragment들이 FragmentManager(아까 배웠죠?)에 박제되어 Activity가 종료되지 않는 한 제거되지 않아요! 그래서 Fragment의 개수가 많아지면 메모리 누수가 발생할 수 있어요!
4. **FragmentStatePagerAdapter**는 범위를 지정할 수 있는데, 범위 밖의 Fragment는 FragmentManager에서 지워주고 Adapter 내부에 저장시켜 둔 뒤 범위 안에 들어왔을 때 재생성 시키고 상태가 복원됩니다!  
즉, Adapter가 유연하게 Fragment를 관리해줍니다!

### FragmentStatePagerAdapter 왜 쓰나요?!



탭을 누를 때,  
Activity 교체 없이  
빨간 박스 UI만 변화한다.

1. 이런 UI를 구성했을 때, Fragment 상태를 보존 시켜 불필요한 서버 통신을 확 줄일 수 있다.
2. 앱 한 몇 십 번 터뜨려봐야 이게 얼마나 용이한지 아는데... 우리는 고생하지 말고 미리 알고 가는 걸로!!!
3. TabLayout과 콜라보를 이뤄서 예쁜 탭 구성 가능

### FragmentStatePagerAdapter 사용법 - 1

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout

xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">

    <android.support.v4.view.ViewPager
        android:id="@+id/vp_main_act_view_pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    </android.support.v4.view.ViewPager>

    <android.support.design.widget.TabLayout
        android:id="@+id/tl_main_act_bottom_tab"
        android:layout_width="match_parent"
        android:layout_height="56dp"
        android:layout_alignParentBottom="true">
    </android.support.design.widget.TabLayout>

</RelativeLayout>
```

- 필요한 View 구성

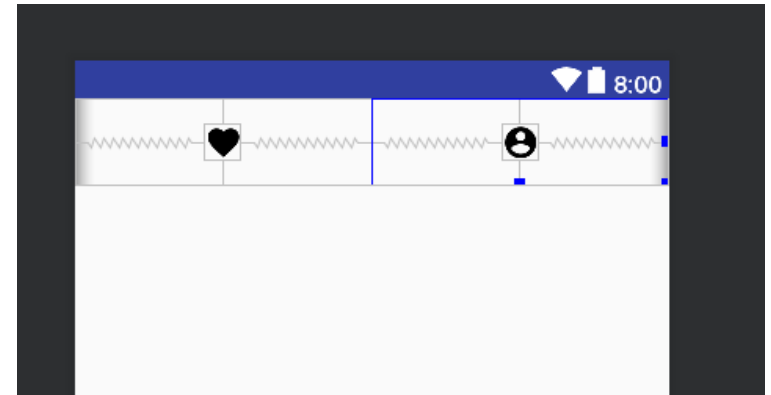
1. Fragment들이 올려질 ViewPager 만들기
2. 바텀 탭 바가 올라갈 TabLayout 만들기

<activity\_main.xml>

### FragmentStatePagerAdapter 사용법 - 2

- 바텀 탭 View를 만들어 봅시다!

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="match_parent"
    android:layout_height="56dp">
    <RelativeLayout
        android:id="@+id/btn_bottom_tab_home"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <ImageView
            android:layout_centerInParent="true"
            android:src="@drawable/ic_favorite_black_24dp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </RelativeLayout>
    <RelativeLayout
        android:id="@+id/btn_bottom_tab_user"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <ImageView
            android:layout_centerInParent="true"
            android:src="@drawable/ic_account_circle_black_24dp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </RelativeLayout>
</LinearLayout>
```





### FragmentStatePagerAdapter 사용법 - 3

- adapter 만들기

```
class MainBottomTabPageAdapter(fm : FragmentManager, private val fragNum : Int) : FragmentStatePagerAdapter(fm){  
    override fun getItem(position: Int): Fragment? {  
        return when(position){  
            0 -> HomeFragment()  
            1 -> UserFragment()  
            else -> null  
        }  
    }  
    override fun getCount(): Int = fragNum  
}
```

### FragmentStatePagerAdapter 사용법 - 4

- 앞서 만든 View들과 adapter를 MainActivity에서 조립해보아요~

```
private fun configureBottomTab(){
    vp_main_act_view_pager.adapter = MainBottomTabPagerAdapter(supportFragmentManager, 2)
    vp_main_act_view_pager.offscreenPageLimit = 2
    tl_main_act_bottom_tab.setupWithViewPager(vp_main_act_view_pager)

    val bottomTabLayout : View = (this.getSystemService(android.content.Context.LAYOUT_INFLATER_SERVICE) as LayoutInflater)
        .inflate(R.layout.bottom_tab_main, null, false)

    tl_main_act_bottom_tab.getTabAt(0)!!.customView = bottomTabLayout.findViewById(R.id.btn_bottom_tab_home) as RelativeLayout
    tl_main_act_bottom_tab.getTabAt(1)!!.customView = bottomTabLayout.findViewById(R.id.btn_bottom_tab_user) as RelativeLayout
}
```

### HomeFragment를 싱글톤으로 만들어 봅시다!

```
companion object {  
    private var instance : HomeFragment? = null  
  
    @Synchronized  
    fun getInstance() : HomeFragment {  
        if (instance == null){  
            instance = HomeFragment()  
        }  
        return instance!!  
    }  
}
```

### Bundle()을 통한 데이터 전송

```
private var myData1: String? = null
private var myData2: String? = null

companion object {
    private var instance : HomeFragment? = null

    @Synchronized
    fun getInstance(data1 : String, data2 : String) : HomeFragment {
        if (instance == null){
            instance = HomeFragment().apply {
                arguments = Bundle().apply {
                    putString("data1", data1)
                    putString("data2", data2)
                }
            }
        }
        return instance!!
    }
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    arguments?.let {
        myData1 = it.getString("data1")
        myData2 = it.getString("data2")
    }
}
```

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

ODo IT SOPT O

THANK U

PPT 디자인 한승미 세미나 자료 배다슬

SHOUT OUR PASSION TOGETHER  
**SOPT**