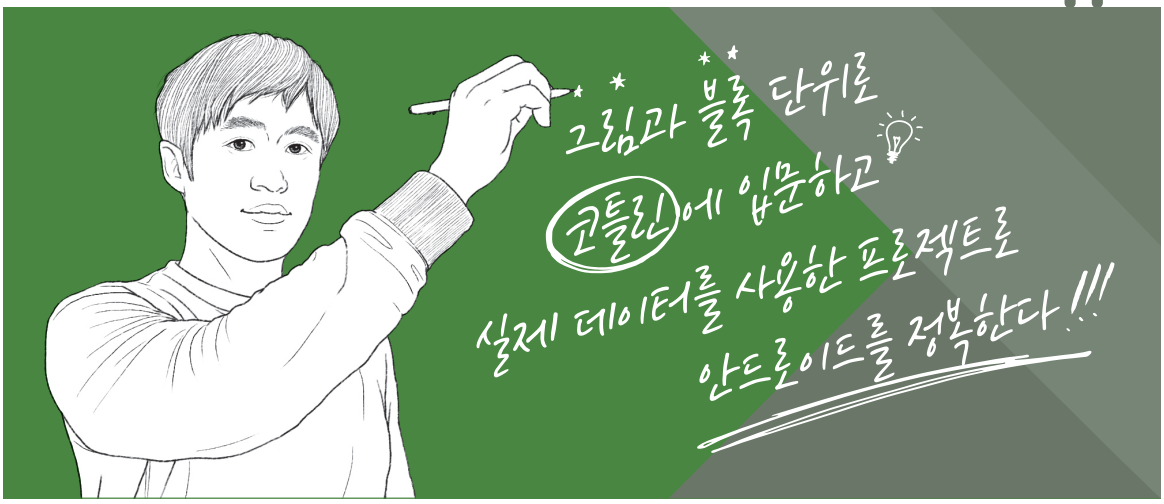
이것이 안드로이드다

with 코틀린

안드로이드 스튜디오 4.x
독자를 위한 PDF 추가 제공



안드로이드 입문의 3가지 장벽, 언어+실전+환경 완벽 대응!

고돈호 지음

안드로이드 10/11 프리뷰 테스트

안드로이드 스튜디오 3.6

동영상 강의 무료 제공

한빛미디어 Hanbit Media, Inc.

Chapter
05

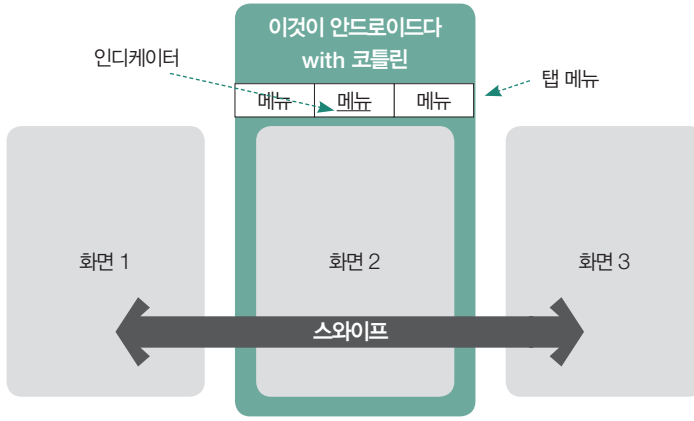
화면 구성하기

안드로이드 스튜디오 4.x 버전에서
뷰페이저(365 ~ 385쪽)가 업데이트되었습니다.

4.x 버전으로 실습할 때는 이 PDF를 활용하세요.

5 탭 메뉴로 화면 구성하기 : ViewPager와 TabLayout

스마트폰에서 가장 많이 사용되는 메뉴의 형태는 탭이나 스와이프로 화면을 전환하는 형태입니다. 다음 그림에서 메뉴를 클릭하면 화면이 전환되고 화면을 좌우로 스와이프하면 화면 전환과 동시에 메뉴의 인디케이터도 함께 동작합니다.



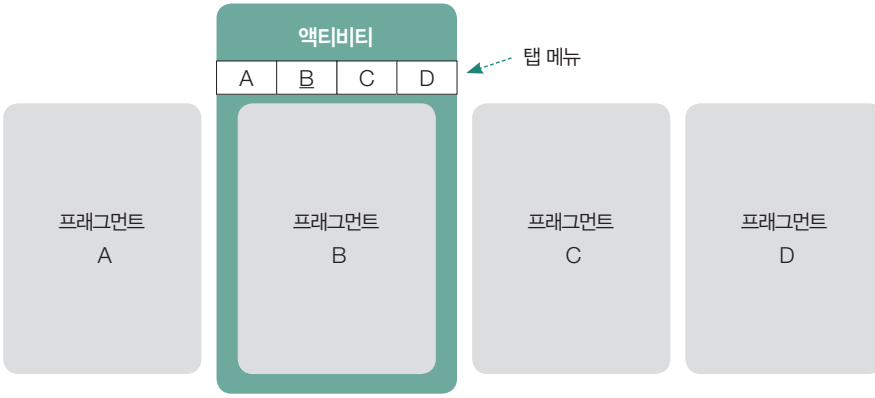
⚠ 스와이프는 손가락으로 화면을 쓸어 넘기는 동작을 의미합니다.

안드로이드에서는 스와이프(Swipe)로 화면을 좌우로 전환할 수 있도록 컨테이너인 뷰페이저(ViewPager)를 제공하고 탭 메뉴 구성을 위해서는 탭 레이아웃(TabLayout)을 제공합니다. 제공되는 2개의 컨테이너를 소스 코드에서 코드로 연결하면 메뉴 화면을 손쉽게 구성할 수 있습니다.

지금부터 뷰페이저와 탭 레이아웃을 사용해서 탭 메뉴와 스와이프로 화면을 전환하는 레이아웃을 구성하고 사용해보겠습니다.

5.1 ViewPager에서 프래그먼트 사용하기

탭 메뉴와 함께 4개의 화면을 프래그먼트로 구성해보겠습니다. 각 프래그먼트에 해당하는 4개의 메뉴를 탭으로 구성한 후에 탭 메뉴를 클릭하거나 또는 스와이프하면 다음 화면으로 전환됩니다.



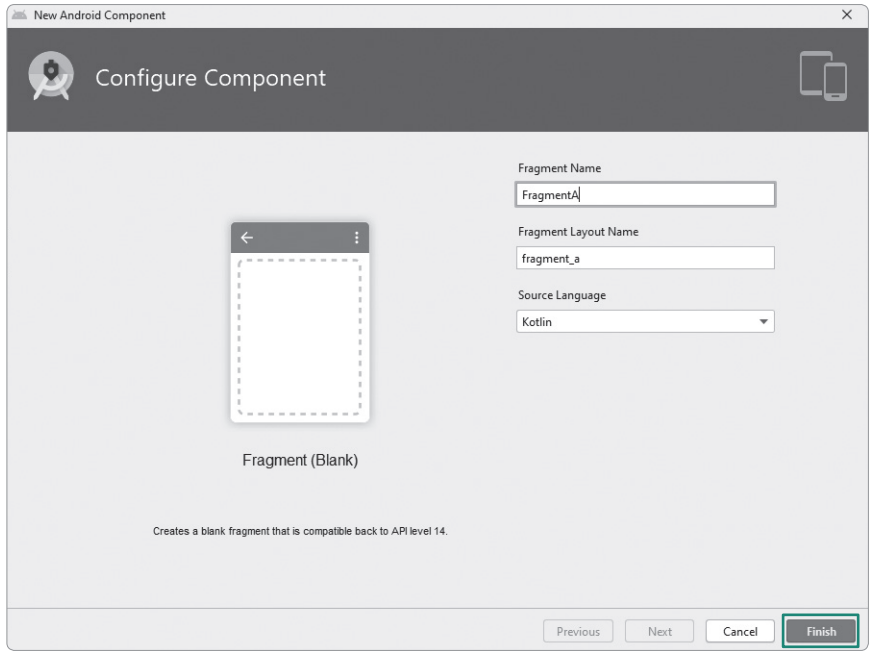
프래그먼트 화면 4개 만들기

새 프로젝트 ViewPagerFragment를 하나 생성합니다. 지금부터 프래그래먼트를 총 4개 만들 텐데 각 프래그먼트의 이름만 'FragmentA', 'FragmentB', 'FragmentC', 'FragmentD'로 다를 뿐 생성하는 방법은 같으니 총 4번 반복해서 만들어봅니다.

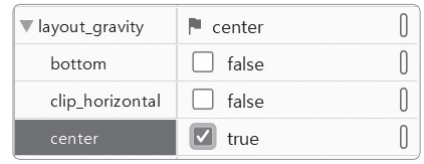
01. 탐색기의 [app]–[java] 디렉터리 아래에 있는 패키지명을 마우스 우클릭하면 나타나는 메뉴에서 [New]–[Fragment]–[Fragment(Blank)]를 선택합니다.

02. 프래그먼트 이름에 'FragmentA'라고 입력합니다. 프래그먼트 레이아웃 이름은 자동으로 생성되는데 fragment_만 있거나 fragment_fragment_a와 같이 이름이 중복되어 있다면 'fragment_a'로 변경합니다. (클래스의 이름을 참조해서 레이아웃 파일의 이름이 결정되는데 fragment_a.xml 형식으로 된 이름을 자동으로 만들기 위해서는 A를 이름 앞에 작성하고 Fragment를 뒤에 붙여서 AFragment라고 하면 됩니다.)

03. [Finish] 버튼을 클릭해서 프래그먼트를 생성합니다.



04. fragment_a.xml 파일을 열고 화면에 기본으로 생성되어 있는 텍스트뷰의 layout_width와 layout_height의 속성을 'wrap_content'로 변경하고 text는 '프래그먼트 A'라고 입력합니다. 프레임 레이아웃에는 정렬 기능이 따로 없기 때문에 텍스트뷰를 선택한 상태에서 layout_gravity 속성값을 'center'로 바꿔주면 가운데 정렬이 됩니다.



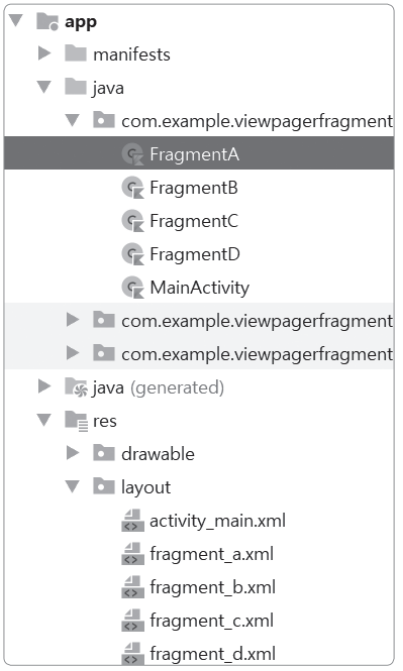
여기서 잠깐

속성 창에서 입력할 수 없어요.

속성 창에서 입력되지 않을 경우 다음처럼 Code 모드에서 직접 속성값을 수정할 수도 있습니다.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="프래그먼트 A" />
```

05. 01~04를 세 번 더 반복해서 4개의 프래그먼트를 모두 만들면 탐색기의 파일 구조가 우측 그림과 같이 됩니다. 액티비티 파일인 MainActivity 1개와 프래그먼트 4개의 레이아웃 파일이 있어야 합니다.



ViewPager와 Adapter 만들기

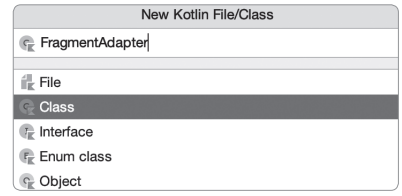
뷰페이저는 리사이클러뷰와 구현 방식이 비슷한데 한 화면에 하나의 아이템만 보여지는 리사이클러뷰라고 생각하면 됩니다. 페이저어댑터를 통해서 뷰페이저에서 보여질 화면들을 연결하는 구조도 리사이클러뷰와 동일합니다.

메인 레이아웃에 뷰페이저를 배치하고 소스 코드에서 연결하겠습니다.

01. activity_main.xml을 열고 가운데 있는 텍스트뷰는 삭제합니다.

02. 팔레트의 컨테이너에 있는 뷰페이저2(안드로이드 스튜디오 3버전의 뷰페이저가 뷰페이저2로 변경)를 드래그해서 추가하고, 상하좌우 컨스트레인트를 화면 가장자리에 연결합니다. id 속성에는 'viewPager'를 입력하고 layout_width와 layout_height의 속성에는 '0dp'를 입력합니다.

⚠ 처음 추가하게 되면 라이브러리를 추가할 것인지 묻는 팝업창이 나타날 수도 있습니다. [OK]를 눌러서 추가합니다.

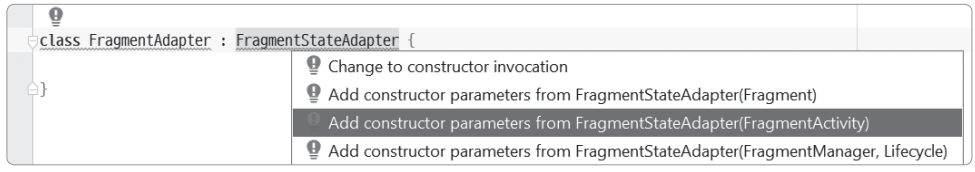


03. 프래그먼트를 뷰페이저에 보여주기 위해서 뷰페이저어댑터를 만들어야 합니다. 마치 리사이클러뷰에서 Adapter를 상속받아 커스텀어댑터를 만들었던 것처럼 프래그먼트를 담을 수 있는 FragmentStateAdapter를 상속받아서 FragmentAdapter를 만들겠습니다. java 디렉터리 밑에 있는 패키지명을 마우스 우클릭한 다음 [New]-[Kotlin File/Class]를 선택합니다. 코틀린 파일 생성 팝업의 입력란에 'FragmentAdapter'를 입력한 후 Class를 더블클릭해서 파일을 생성합니다.

04. 생성된 클래스 파일에서 FragmentStateAdapter를 상속받도록 코드를 수정합니다. 끝에 (괄호)를 생략하고 상속받습니다.

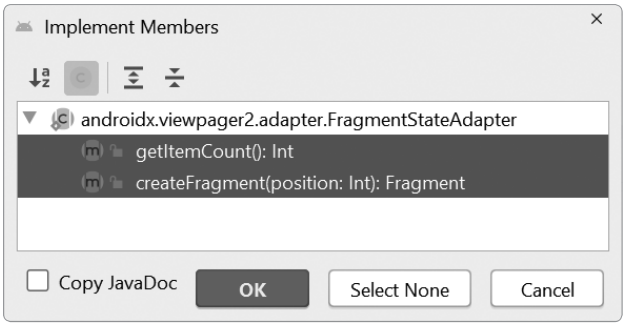
```
class FragmentAdapter : FragmentStateAdapter {
}
```

05. FragmentStateAdapter 아래에 빨간 밑줄이 생기는데 글자를 클릭 후 Alt + Enter 키를 입력해서 나타나는 팝업창에서 [Add constructor ... (FragmentActivity)]를 선택해서 생성자를 추가합니다.



```
class FragmentAdapter((fragmentActivity: FragmentActivity) : FragmentStateAdapter(frag
mentActivity) {

}
```

06. 여전히 클래스 이름 아래에 빨간색 밑줄이 생기는데 클릭한 후에 키보드의 Alt + Enter 키를 누르고 나타나는 메뉴에서 [Implement members]를 선택한 다음 나오는 메서드 목록에서 모두 선택하고 [OK]를 클릭하면 코드가 자동 생성됩니다. TODO 메서드는 모두 삭제합니다.




```
class FragmentAdapter(fragmentActivity: FragmentActivity) :
                      FragmentStateAdapter(fragmentActivity) {
    override fun getItemCount(): Int {
        // TODO("Not yet implemented")
    }

    override fun createFragment(position: Int): Fragment {
        TODO("Not yet implemented")
    }
}
```

여기서 잠깐

FragmentStateAdapter의 필수 메서드

- createFragment(): 현재 페이지의 position이 파라미터로 넘어옵니다. position에 해당하는 위치의 프래그먼트를 만들어서 안드로이드에 반환해야 합니다.
- getItemCount(): 어댑터가 화면에 보여줄 전체 프래그먼트의 개수를 반환해야 합니다.

07. 리사이클러뷰 어댑터에서 사용했던 것처럼 페이저어댑터도 화면에 표시해줄 아이템의 목록이 필요합니다. `fragmentList` 변수를 하나 만들고 초기화합니다.

```
var fragmentList = listOf<Fragment>()
```

Fragment는 androidx.fragment.app 패키지에 있는 Fragment를 import합니다.

⚠ 메뉴 형태로 사용하는 뷰페이저의 화면 아이템은 대부분 중간에 개수가 늘거나 줄지 않고, 처음에 정해진 개수만큼 사용합니다. 그래서 mutableListOf가 아닌 listOf를 사용하는 것이 효율적입니다.

08. 앞에서 `implement`했던 메서드를 마저 구현합니다. 먼저 페이지의 개수를 결정하기 위해, `getItemCount()` 메서드에서 프래그먼트의 개수를 리턴합니다.

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

09. 페이지가 요청될 때 `getItem()`으로 요청되는 페이지의 `position` 값이 넘어옵니다. `position` 값을 이용해서 프래그먼트 목록에서 해당 `position`에 있는 프래그먼트 1개를 리턴합니다. 다음은 `createFragment()` 메서드를 포함한 전체 코드입니다.

```
package com.example.viewpagerfragment

import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentActivity
import androidx.viewpager2.adapter.FragmentStateAdapter

class FragmentAdapter(fragmentActivity: FragmentActivity) : FragmentStateAdapter(fragm
entActivity) {
```

```
    var fragmentList = listOf<Fragment>()

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

    override fun createFragment(position: Int): Fragment {
        return fragmentList[position]
    }
}
```

메인 액티비티에서 연결하기

지금까지 만든 프래그먼트와 어댑터를 메인 액티비티의 소스 코드에서 연결합니다.

01. MainActivity.kt의 onCreate() 메서드 안에 프래그먼트 목록을 생성하는 코드를 추가합니다.

```
val fragmentList = listOf(FragmentA(), FragmentB(), FragmentC(), FragmentD())
```

02. adapter를 생성하고 앞에서 생성해 둔 프래그먼트 목록을 저장합니다. adapter의 첫 번째 파라미터에는 항상 supportFragmentManger를 사용합니다.

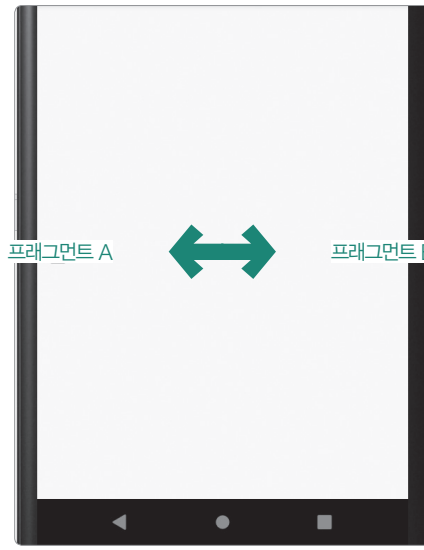
```
val adapter = FragmentAdapter(this)
adapter.fragmentList = fragmentList
```

03. 레이아웃의 viewPager를 import하고 어댑터를 적용합니다. 다음은 MainActivity.kt의 전체 코드입니다.

```
package com.example.viewpagerfragment

import android.os.Bundle
import androidx.fragment.app.FragmentActivity
import com.google.android.material.tabs.TabLayoutMediator
import kotlinx.android.synthetic.main.activity_main.*
```

```
// ViewPager 2
class MainActivity : FragmentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val fragmentList = listOf(FragmentA(), FragmentB(), FragmentC(), FragmentD())
        val adapter = FragmentAdapter(this)
        adapter.fragmentList = fragmentList
        viewPager.adapter = adapter
    }
}
```

04. 작성한 코드를 실행합니다. 화면을 양옆으로 스와이프해보면 프래그먼트 A부터 D까지 화면이 이동하는 것을 확인할 수 있습니다.

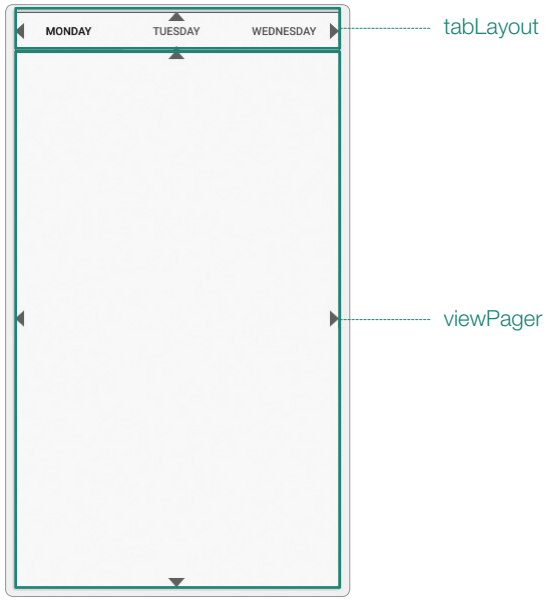


뷰페이저를 사용하면 여러 개의 화면이 스와이프 되는 앱을 짧은 코드로 작성할 수 있습니다.

TabLayout 적용하기

앞에서 만든 화면에 4개의 탭 메뉴를 상단에 배치하고 탭 메뉴 클릭 시 해당하는 프래그먼트로 이동하는 코드를 작성하겠습니다. 먼저 레이아웃에 탭 메뉴를 삽입하겠습니다.

01. activity_main.xml을 열고 팔레트의 컨테이너 카테고리에 있는 탭 레이아웃을 드래그해서 뷰페이저 위에 배치합니다. 뷰페이저의 위쪽 Constraint를 삭제한 후 작업하는 것이 편합니다. TabLayout이 정상적으로 배치되었으면 뷰페이저의 위쪽 Constraint를 TabLayout 아래에 연결해서 그림과 같이 만들어 줍니다.



02. 3버전의 뷰페이저와는 다르게 4버전의 뷰페이저2에서는 TabLayoutMediator를 사용해서 TabLayout과 뷰페이저를 연결합니다. 먼저 메뉴명으로 사용할 이름들을 배열에 저장합니다(MainActivity의 onCreate() 함수 마지막 줄에 작성합니다).

```
val tabTitles = listOf<String>("A", "B", "C", "D")
```

03. TabLayoutMediator를 사용해서 TabLayout과 뷰페이저를 연결합니다. 코드블럭으로 전달되는 tab 파라미터의 text속성에 앞에서 미리 정의해둔 메뉴명을 입력합니다.

```
TabLayoutMediator(tabLayout, viewPager) { tab, position ->
    tab.text = tabTitles[position]
}.attach()
```

04. 코드 블럭의 끝에서 attach() 함수를 호출해서 적용합니다.

```
package com.example.viewpagerfragment

import android.os.Bundle
import androidx.fragment.app.FragmentActivity
import com.google.android.material.tabs.TabLayoutMediator
import kotlinx.android.synthetic.main.activity_main.*

// ViewPager 2
class MainActivity : FragmentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val fragmentList = listOf(FragmentA(), FragmentB(), FragmentC(), FragmentD())
        val adapter = FragmentAdapter(this)
        adapter.fragmentList = fragmentList
        viewPager.adapter = adapter

        val tabTitles = listOf<String>("A", "B", "C", "D")
        TabLayoutMediator(tabLayout, viewPager) { tab, position ->
            tab.text = tabTitles[position]
        }.attach()
    }
}
```

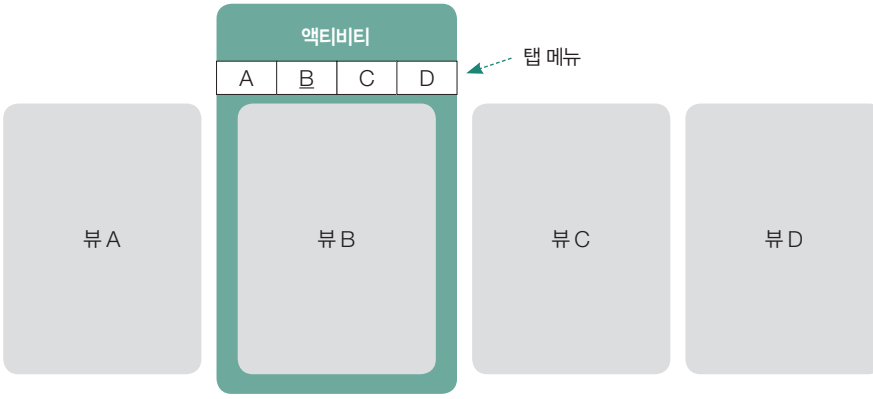
05. 에뮬레이터에서 실행하면 메뉴와 뷰페이저가 모두 정상적으로 동작합니다.

5.2 View를 사용하는 뷰페이저 만들기

앞에서 프래그먼트를 사용해 뷰페이저를 구현했는데 이 방식은 각각의 화면들이 독립적으로 구성될 필요가 있을 때 사용할 수 있습니다. 그런데 리사이클러뷰에서처럼 하나의 아이템레이아웃을 사용해서 반복적으로 동일한 구조의 텍스트나 이미지를 보여주는 용도라면 프래그먼트보다는 뷰를 사용합니다.

뷰는 목록을 가로로 스와이프해서 보여줄 필요가 있을 때 사용하는데 일반적인 사진 갤러리 앱이 동작하는 방식을 생각하면 됩니다.

프래그먼트 대신에 이번에는 뷰를 사용해서 각각의 메뉴 화면을 다음의 그림처럼 구성하겠습니다.



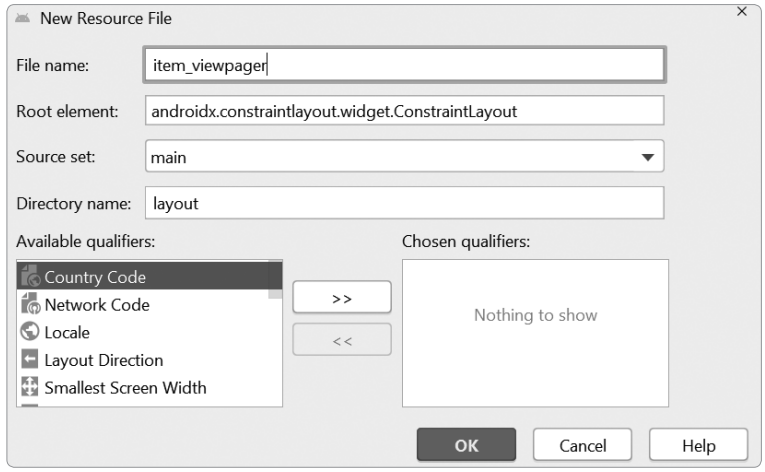
ViewPagerView 프로젝트를 하나 새로 생성합니다.

아이템 레이아웃 만들기

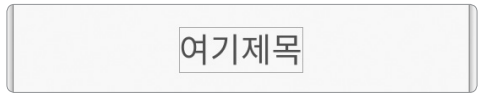
먼저 아이템 레이아웃을 만들겠습니다.

01. 리사이클러뷰의 아이템 레이아웃처럼 하나의 뷰에서 사용할 아이템 레이아웃을 생성합니다. [res]-[layout] 디렉터리를 마우스 우클릭하면 나타나는 메뉴에서 [New]-[Layout Resource file]을 선택합니다.

02. File name에 'item_viewpager'라고 입력하고 파일을 생성합니다. 프래그먼트와는 다르게 레이아웃 파일을 먼저 생성한 후에 클래스 파일을 생성합니다.



03. 레이아웃 파일 가운데에 텍스트뷰를 하나 가져다 놓고 텍스트뷰의 text 속성을 '여기제목'으로 입력합니다. 텍스트뷰의 id에 textView를 입력합니다. **02**에서 변경한 게 없으면 최상위 레이아웃은 ConstraintLayout입니다. 컨스트레인트는 네 방향 모두 연결해서 가운데에 오도록 배치합니다.

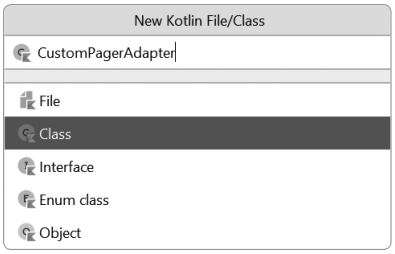


⚠ 프래그먼트와는 다르게 레이아웃 파일만 생성하고 따로 클래스는 만들지 않습니다.

CustomPagerAdapter 만들기

앞에서 생성한 레이아웃을 사용하는 커스텀 아답터를 생성합니다. 목록을 만들 때 사용하는 RecyclerView.Adapter를 상속받아서 사용합니다.

01. CustomPagerAdapter 클래스를 하나 생성합니다. 이후부터는 리사이클러뷰를 사용하는 방법과 동일합니다.



⚠ 뷰페이저에 리사이클러뷰 아답터를 사용하면 기존에 세로로 출력되는 것을 가로로 출력되도록 해준다고 생각하면 이해하기가 더 쉽습니다.

02. 먼저 RecyclerView.ViewHolder를 상속받는 Holder 클래스를 파일 아래에 하나 만듭니다. Holder 클래스의 itemView파라미터로 우리가 앞에서 미리 만들어둔 item_viewpager 레이아웃이 전달됩니다.

```
class CustomPagerAdapter{
}

class Holder(itemView: View) : RecyclerView.ViewHolder(itemView) {

}
```

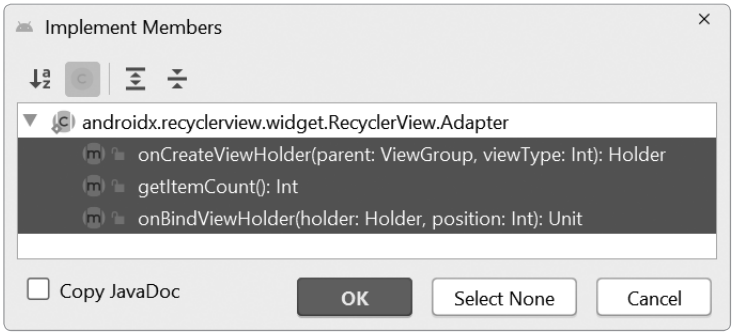
03. Holder 클래스 안에 setText() 함수를 하나 만들고 item_viewpager 레이아웃 안에 미리 만들어둔 텍스트뷰(id:textView)에 값을 입력하는 코드를 작성합니다. setText() 함수의 파라미터에는 가상으로 text:String이라고 미리 정의하고 사용합니다.

```
class Holder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    fun setText(text:String) {
        itemView.textView.text = text
    }
}
```

04. CustomPagerAdapter에서 RecyclerView.Adapter를 상속받고 제네릭으로 앞에서 만든 Holder클래스를 지정합니다.

```
class CustomPagerAdapter : RecyclerView.Adapter<Holder>(){
}
```

05. 클래스 안쪽을 클릭한 상태로 키보드의 Ctrl + I 를 입력하면 나타나는 창에서 세 개의 함수를 선택하고 오버라이드합니다.




```
class CustomPagerAdapter : RecyclerView.Adapter<Holder>(){
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): Holder {
        TODO("Not yet implemented")
    }

    override fun getItemCount(): Int {
        // TODO("Not yet implemented")
    }

    override fun onBindViewHolder(holder: Holder, position: Int) {
        // TODO("Not yet implemented")
    }
}
```

06. 아답터에서 사용 할 textList 변수를 선언하고 listOf 함수로 초기화합니다. MainActivity에서 아답터를 생성한 후 textList변수로 각각의 페이지에서 보여줄 텍스트를 전달합니다.

```
var textList = listOf<String>()
```

07. getItemCount 함수는 몇 개의 페이지가 보여질 건지 결정합니다.

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

08. onCreateViewHolder()에서 item_viewpager를 inflate한 후 Holder에 담아서 안드로이드에 전달합니다.

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): Holder {
    val view = LayoutInflater.from(parent.context).inflate(R.layout.item_viewpager,
parent, false)
    return Holder(view)
}
```

09. 마지막으로 onBindViewHolder()에서 Holder 에 만들어둔 setText 함수를 호출해서 화면에 표시합니다.

```
override fun onBindViewHolder(holder: Holder, position: Int) {
    val text = textList[position]
    holder.setText(text)
}
```

다음은 CustomPagerAdapter의 전체 코드입니다.

```
package com.example.viewpagerview

import android.view.LayoutInflater
```

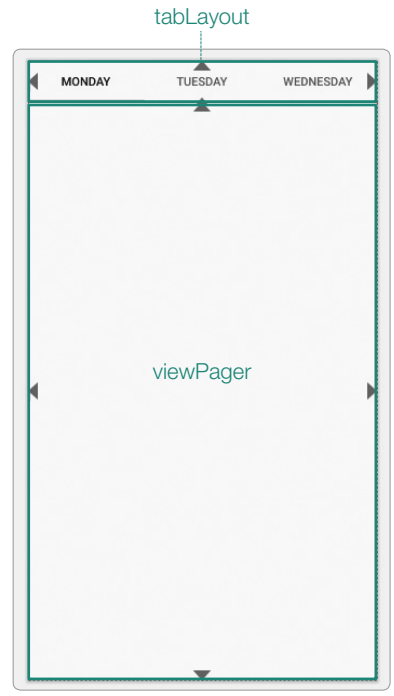
```
import android.view.View
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import kotlinx.android.synthetic.main.item_viewpager.view.*

class CustomPagerAdapter : RecyclerView.Adapter<Holder>(){
    var textList = listOf<String>()

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): Holder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.item_viewpager,
parent, false)
        return Holder(view)
    }

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

    override fun onBindViewHolder(holder: Holder, position: Int) {
        val text = textList[position]
        holder.setText(text)
    }
}

class Holder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    fun setText(text:String) {
        itemView.textView.text = text
    }
}
```

레이아웃 파일에 ViewPager2와 TabLayout 추가하고 소스코드 연결하기

앞에서 만든 아답터를 연결할 화면을 작성합니다. 프래그먼트에서 작성했던 것과 동일합니다.

01. activity_main.xml 파일을 열고 화면에 있는 텍스트뷰는 삭제하고, 팔레트에서 TabLayout을 드래그해서 화면에 추가합니다. 좌, 우와 위쪽 Constraint를 연결한 후 id에 tabLayout을 입력합니다.

02. ViewPager2를 드래그해서 tabLayout아래에 배치하고 상,하,좌,우 컨스트레인트를 연결합니다.



03. 이제 코드를 연결하겠습니다. MainActivity 파일을 열고 onCreate함수에 코드를 추가합니다. 뷰페이저에서 사용할 데이터를 가상으로 생성한 후 textList 변수에 담습니다.

```
val textList = listOf("뷰A", "뷰B", "뷰C", "뷰D")
```

04. 커스텀 아답터를 생성합니다.

```
val adapter = CustomPagerAdapter()
```

05. 생성해둔 가상 데이터를 아답터에 전달합니다.

```
adapter.textList = textList
```

06. viewPager에 아답터를 연결합니다.

```
viewPager.adapter = adapter
```

07. 메뉴명으로 사용할 이름을 배열에 저장합니다.

```
val tabTitles = listOf("View A", "View B", "View C", "View D")
```

08. TabLayoutMediator를 사용해서 TabLayout과 뷰페이저를 연결합니다. 코드블럭으로 전달되는 tab 파라미터의 text 속성에 앞에서 미리 정의해둔 메뉴명을 입력합니다. 코드 블럭의 끝에서 attach() 함수를 호출해서 적용합니다.

```
TabLayoutMediator(tabLayout, viewPager) { tab, position ->
    tab.text = tabTitles[position]
}.attach()
```

09. 에뮬레이터에서 실행하고 확인합니다. 다음은 아래는 MainActivity의 전체 코드입니다.

```
package com.example.viewpagerview

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import com.google.android.material.tabs.TabLayoutMediator
import kotlinx.android.synthetic.main.activity_main.*

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

        val textList = listOf("뷰A", "뷰B", "뷰C", "뷰D")
        val adapter = CustomPagerAdapter()
        adapter.textList = textList
        viewPager.adapter = adapter

        val tabTitles = listOf("View A", "View B", "View C", "View D")
        TabLayoutMediator(tabLayout, viewPager) {tab, position ->
            tab.text = tabTitles[position]
        }.attach()
    }
}
```

여기서 잠깐

Ctrl + I 키와 Ctrl + O 키의 차이

- Ctrl + I (Implement): 메서드명만 있는 인터페이스가 설계되어 있습니다. 메서드 내부에 코드를 작성해두면 부모 클래스에 작성되어 있는 코드에서 우리가 작성한 인터페이스 메서드를 호출해서 사용합니다. 인터페이스는 구현하지 않으면 컴파일되지 않습니다.
- Ctrl + O (Override): 부모 클래스에 이미 만들어져 있는 메서드를 내 코드에 맞게 재정의하는 것입니다. 구현하지 않아도 컴파일되며, 부모 클래스에 있는 메서드가 호출되고 실행됩니다.