세상의 속도를
따라잡고 싶다면

Do it!

깡샘의 안드로이드 앱 프로그래밍 with 코틀린

이지스퍼블리싱(주)

11

제트팩 라이브러리

이지스퍼블리싱(주)

11-1 제트팩과 androidx 소개

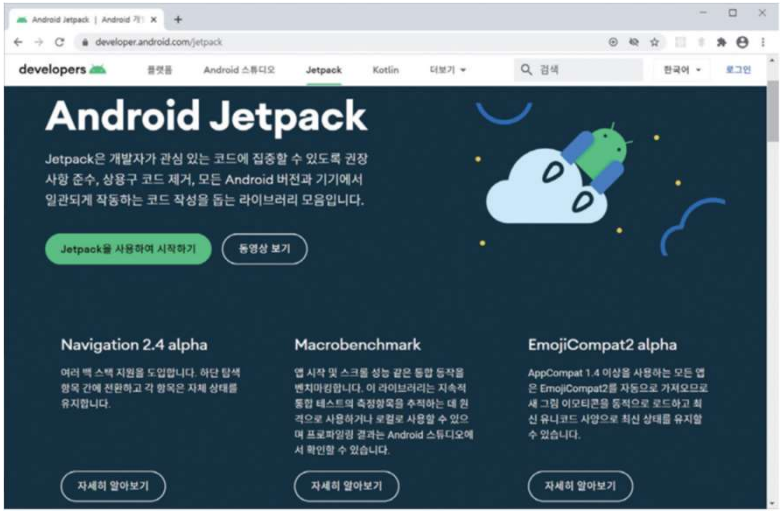
플랫폼 API

- 플랫폼 API는 ART에서 제공하는 안드로이드 라이브러리

11-1 제트팩과 androidx 소개

제트팩

- 제트팩은 구글에서 안드로이드 앱 개발용으로 제공하는 다양한 라이브러리 모음
- androidx로 시작하는 패키지명을 사용
  - 앱을 개발하는 데 필요한 권장 아키텍처를 제공합니다.
  - API 레벨의 호환성 문제를 해결합니다.
  - 플랫폼 API에서 제공하지 않는 다양한 기능을 제공합니다.



11-1 제트팩과 androidx 소개

androidx 라이브러리

- 화면 구성과 관련된 라이브러리
  - androidx.appcompat: 앱의 API 레벨 호환성을 해결합니다.
  - androidx.recyclerview: 목록 화면을 구성합니다.
  - androidx.viewpager2: 스와이프로 넘기는 화면을 구성합니다.
  - androidx.fragment: 액티비티처럼 동작하는 뷰를 제공합니다.
  - androidx.drawerlayout: 옆에서 서랍처럼 열리는 화면을 구성합니다.

11-2 appcompat 라이브러리

- appcompat 라이브러리는 안드로이드 앱의 화면을 구성하는 액티비티를 만들며 API 레벨의 호환성 문제를 해결

• appcompat 라이브러리 선언

```
implementation 'androidx.appcompat:appcompat:1.3.1'
```

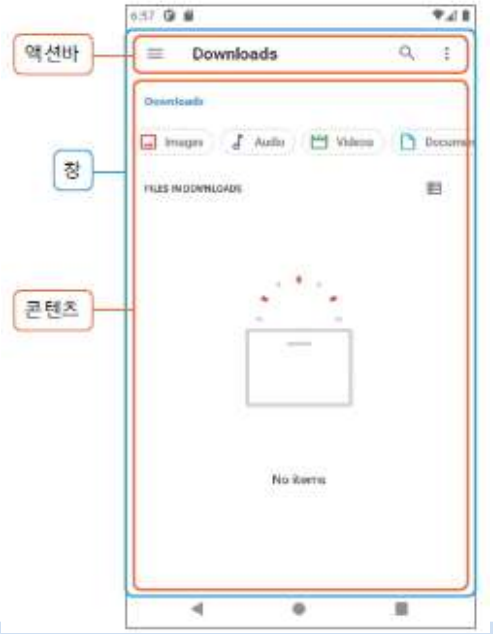
• appcompat 라이브러리 사용

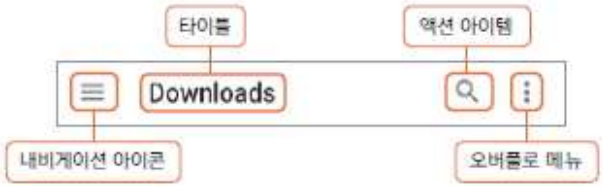
```
import androidx.appcompat.app.AppCompatActivity
(... 생략 ...)
class MainActivity : AppCompatActivity() {
}
```

11-2 appcompat 라이브러리

액션바

- 화면 위쪽에 타이틀 문자열이 출력되는 영역
- 내비게이션 아이콘, 액션 아이템, 오버플로 메뉴 등 다양한 요소를 액션바에 출력할 수 있습니다.





11-2 appcompat 라이브러리

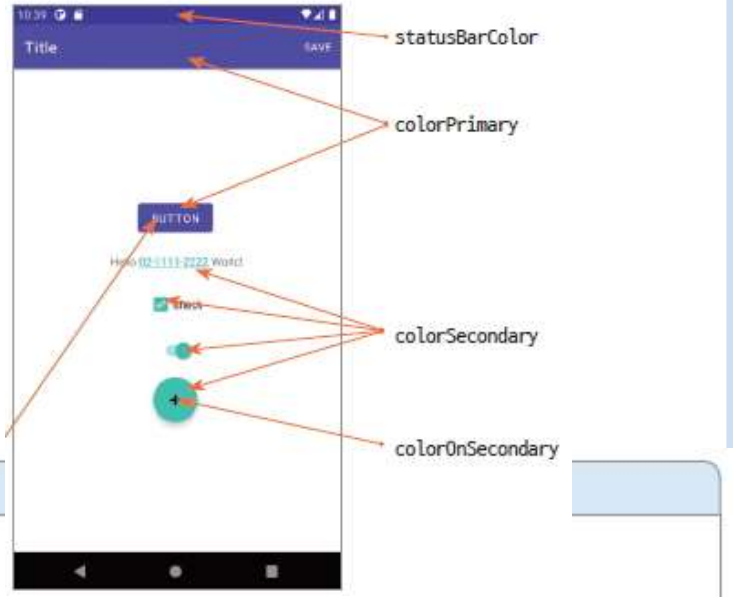
- 액션바 색상 설정
  - 테마 스타일은 res/values 디렉터리에 있는 themes.xml 파일에 선언

• 기본으로 작성된 테마 파일의 내용

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- 기본 테마 -->
    <style name="Theme.AndroidLab" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- 주 색상 -->
        <item name="colorPrimary">@color/purple_500</item>
        <item name="colorPrimaryVariant">@color/purple_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- 부 색상 -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item
        <item name="colorOnSecondary">@color/black</item>
        <!-- 상태바 색상 -->
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
        <!-- 기본 테마 설정 -->
    </style>
</resources>
```

• 매니페스트 파일의 테마 설정

```
<application
    (... 생략 ...)
    android:theme="@style/Theme.AndroidLab">
```



11-2 appcompat 라이브러리

- 액션바 숨기기 설정
  - Theme.MaterialComponents.DayNight.NoActionBar를 상속받으면 액션바가 나오지 않습니다.

• 액션바 숨기기

```
<style name="Theme.AndroidLab" parent="Theme.MaterialComponents.DayNight.NoActionBar">
    (... 생략 ...)
</style>
```

• <item> 속성으로 숨기기

```
<style name="Theme.AndroidLab"
    parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    (... 생략 ...)
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
</style>
```

11-2 appcompat 라이브러리

- 업 버튼 설정
  - 액션바 왼쪽에 이전 화면으로 되돌아가는 화살표 모양(←)의 업 버튼을 제공할 수 있습니다.

• 매니페스트 파일에서 업 버튼 설정

```
<activity
    android:name=".TwoActivity"
    android:parentActivityName=".MainActivity"></activity>
```

• 업 버튼 클릭 시 자동으로 호출되는 함수 재정의

```
override fun onSupportNavigateUp(): Boolean {
    Log.d("kkang","onSupportNavigateUp")
    return super.onSupportNavigateUp()
}
```

• 액티비티 코드에서 업 버튼 생성

```
class TwoActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        (... 생략 ...)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
    }
    override fun onSupportNavigateUp(): Boolean {
        Log.d("kkang","onSupportNavigateUp")
        onBackPressed()
        return super.onSupportNavigateUp()
    }
}
```

11-2 appcompat 라이브러리

메뉴 구성

- 메뉴는 액션바의 중요한 구성 요소로 액티비티 화면에서 사용자 이벤트를 사용할 수 있도록 합니다.
- 액티비티에 메뉴를 추가하고 싶다면 onCreateOptionsMenu()와 onPrepareOptionsMenu() 함수를 이용
- onCreateOptionsMenu() 함수는 액티비티가 실행되면서 처음에 한 번만 호출
- onPrepareOptionsMenu() 함수는 액티비티가 실행되면서 한 번 호출된 후 오버플로 메뉴가 나타날 때마다 반복해서 호출

11-2 appcompat 라이브러리

• 메뉴 구성 함수

```
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    val menuItem1: MenuItem? = menu?.add(0, 0, 0, "menu1")
    val menuItem2: MenuItem? = menu?.add(0, 1, 0, "menu2")
    return super.onCreateOptionsMenu(menu)
}
```

▶ 실행 결과

11-2 appcompat 라이브러리

- 메뉴를 사용자가 선택했을 때의 이벤트 처리는 onOptionsItemSelected() 함수를 이용
- 함수의 매개변수는 이벤트가 발생한 메뉴 객체인 MenuItem

• 메뉴 선택 시 이벤트 처리

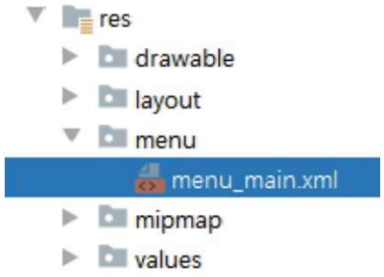
```
override fun onOptionsItemSelected(item: MenuItem): Boolean = when (item.itemId) {
    0 -> {
        Log.d("kkang", "menu1 click")
        true
    }
    1-> {
        Log.d("kkang", "menu2 click")
        true
    }
    else -> super.onOptionsItemSelected(item)
}
```

11-2 appcompat 라이브러리

- 리소스로 메뉴 구현
  - 메뉴를 구성하는 XML 파일은 res 폴더 아래 menu 디렉터리에 만듭니다.

• 메뉴 XML

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/menu1"
        android:title="menu1" />
    <item
        android:id="@+id/menu2"
        android:icon="@android:drawable/ic_menu_add"
        android:title="menu2"
        app:showAsAction="always" />
    <item
        android:id="@+id/menu3"
        android:icon="@android:drawable/ic_menu_search"
        android:title="menu2"
        app:showAsAction="ifRoom" />
</menu>
```



11-2 appcompat 라이브러리

- 액션바에 아이콘으로 나타나게 하려면 showAsAction 속성을 이용
- never(기본): 항상 오버플로 메뉴로 출력합니다.
- ifRoom: 만약 액션바에 공간이 있다면 액션 아이템으로, 없다면 오버플로 메뉴로 출력합니다.
- always: 항상 액션 아이템으로 출력합니다.

• 액티비티 코드에 메뉴 XML 적용

```
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    menuInflater.inflate(R.menu.menu_main, menu)
    return super.onCreateOptionsMenu(menu)
}
```

▶ 실행 결과

11-2 appcompat 라이브러리

- 액션 뷰 이용
  - 액션 뷰는 액션바에서 특별한 기능을 제공하며 대표적으로 androidx.appcompat.widget.SearchView가 있습니다.

• 서치 뷰 사용

```
<item android:id="@+id/menu_search"
    android:title="search"
    app:showAsAction="always"
    app:actionViewClass="androidx.appcompat.widget.SearchView" />
```

11-2 appcompat 라이브러리

- SearchView를 등록한 MenuItem 객체를 얻고 MenuItem 객체에 등록된 SearchView 객체를 구하면 됩니다.

• 서치 뷰 검색 기능 구현

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    val inflater = menuInflater
    inflater.inflate(R.menu.menu_main, menu)
    val menuItem = menu?.findItem(R.id.menu_search)
    val searchView = menuItem?.actionView as SearchView
    searchView.setOnQueryTextListener(object: SearchView.OnQueryTextListener {
        override fun onQueryTextChange(newText: String?): Boolean {
            // 검색어 변경 이벤트

            return true
        }
        override fun onQueryTextSubmit(query: String?): Boolean {
            // 키보드의 검색 버튼을 클릭한 순간의 이벤트
            return true
        }
    })
    return true
}
```

11-2 appcompat 라이브러리

툴바

- 액션바는 액티비티 창이 자동으로 출력하는 액티비티의 구성 요소지만, 툴바는 개발자가 직접 제어하는 뷰라는 데 차이점

11-2 appcompat 라이브러리

- 레이아웃 XML 파일에 다음처럼 Toolbar를 등록

• 레이아웃 XML에 툴바 등록

```
<androidx.appcompat.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    style="@style/Widget.MaterialComponents.Toolbar.Primary" />
```

• 액션바의 내용을 툴바에 적용

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        (... 생략 ...)
        setSupportActionBar(binding.toolbar)
    }
}
```

11-2 appcompat 라이브러리

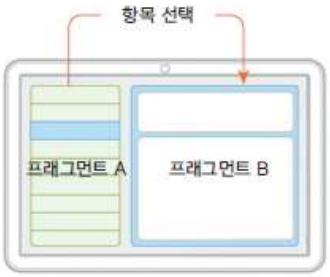
호환성을 고려한 기본 뷰 클래스

- 플랫폼 API에서 제공하는 기본 뷰를 appcompat 라이브러리에서도 제공
- TextView 클래스를 appcompat 라이브러리에서는 App CompatTextView라는 클래스명으로 제공
- AppCompatImageView, AppCom patEditText, AppCompatButton, AppCompatCheckBox, AppCompatRadioButton 등 기본 뷰에 해당하는 뷰를 제공

11-3 프래그먼트 – 액티비티처럼 동작하는 뷰

프래그먼트 소개

- 프래그먼트가 다른 뷰와 다른 점은 액티비티처럼 동작한다는 것
- 액티비티에 구현되는 모든 내용은 프래그먼트 클래스에도 작성할 수 있습니다.



11-3 프래그먼트 – 액티비티처럼 동작하는 뷰

프래그먼트 구현

- 프래그먼트는 androidx.fragment 라이브러리에서 제공
- 프래그먼트는 Fragment를 상속받아 작성하는 클래스
- 최소한으로 작성해야 하는 함수는 onCreateView()
- 이 함수가 자동 호출되며 반환한 View 객체가 화면에 출력

• 프래그먼트 구현

```
(... 생략 ...)
import androidx.fragment.app.Fragment
class OneFragment : Fragment() {
    lateinit var binding: FragmentOneBinding
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = FragmentOneBinding.inflate(inflater, container, false)
        return binding.root
    }
}
```

11-3 프래그먼트 – 액티비티처럼 동작하는 뷰

- 액티비티의 레이아웃 XML에 등록하여 프래그먼트 출력
  - <fragment> 태그로 액티비티 화면에 프래그먼트를 출력
  - <fragment> 태그의 name 속성에 프래그먼트 클래스를 지정

• 프래그먼트 출력

```
<fragment
    android:name="com.example.test11.OneFragment"
    android:id="@+id/fragmentView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

11-3 프래그먼트 – 액티비티처럼 동작하는 뷰

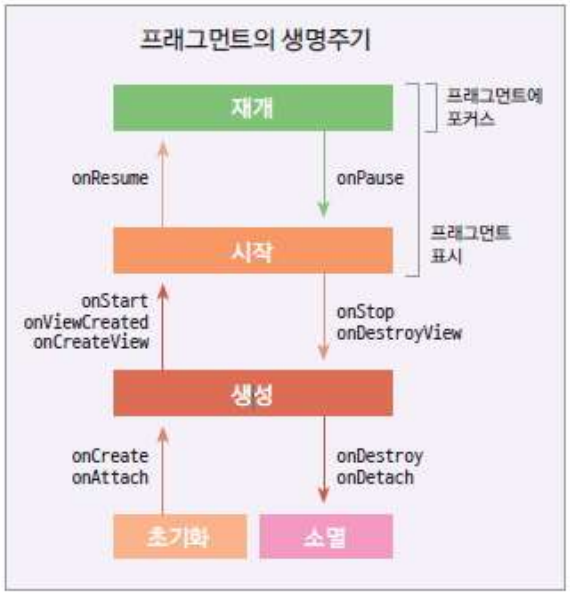
- 액티비티 코드에서 프래그먼트 출력
  - 코드에서 프래그먼트를 동적으로 제어(추가, 제거 등)하려면 FragmentManager로 만든 FragmentTransaction 클래스의 함수를 이용
  - add(int containerViewId, Fragment fragment): 새로운 프래그먼트를 화면에 추가합니다.
  - replace(int containerViewId, Fragment fragment): 추가된 프래그먼트를 대체합니다.
  - remove(Fragment fragment): 추가된 프래그먼트를 제거합니다.
  - commit(): 화면에 적용합니다.

• 프래그먼트 동적 제어

```
val fragmentManager: FragmentManager = supportFragmentManager
val transaction: FragmentTransaction = fragmentManager.beginTransaction()
val fragment = OneFragment()
transaction.add(R.id.fragment_content, fragment)
transaction.commit()
```

11-3 프래그먼트 – 액티비티처럼 동작하는 뷰

- 프래그먼트 생명주기
  - 액티비티의 생명주기 함수인 onCreate(), onStart(), onResume(), onPause(), onStop(), onDestory()를 그대로 가지고 있으며 호출되는 시점도 액티비티와 같습니다.
  - 초기화(initialized), 생성(created), 시작(started), 재개(resumed), 소멸(destroyed) 단계로 구분



11-3 프래그먼트 – 액티비티처럼 동작하는 뷰

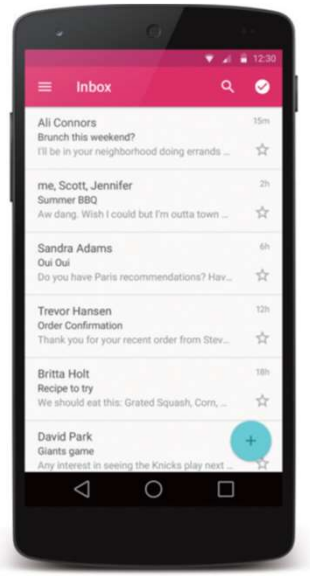
- 백 스택은 프래그먼트가 화면에 보이지 않는 순간 제거하지 않고 저장했다가 다시 이용할 수 있는 기능을 말합니다.
- 백 스택을 사용하지 않으면 프래그먼트가 교체될 때 기존의 프래그먼트는 onDetory까지 호출되어 제거됩니다.
- 백 스택을 사용하면 프래그먼트가 제거되지 않고 onDestoryView 함수까지만 호출됩니다.

• 백 스택 사용 설정

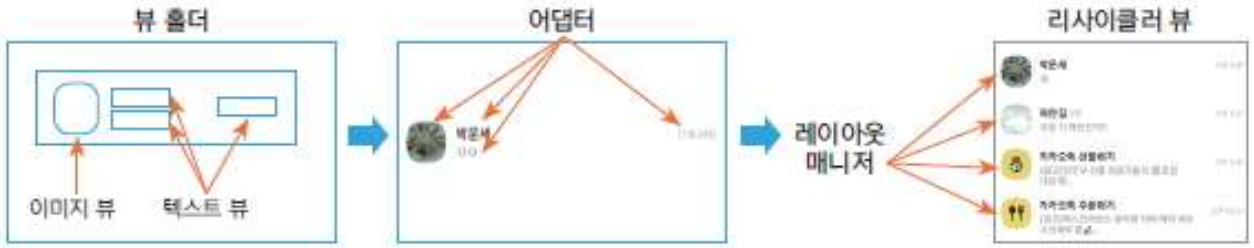
```
transaction.addToBackStack(null)
```

11-4 리사이클러 뷰 – 목록 화면 구성

리사이클러 뷰 기초 사용법



- 구성 요소
  - ViewHolder(필수): 항목에 필요한 뷰 객체를 가집니다.
  - Adapter(필수): 항목을 구성합니다.
  - LayoutManager(필수): 항목을 배치합니다.
  - ItemDecoration(옵션): 항목을 꾸밉니다.



11-4 리사이클러 뷰 – 목록 화면 구성

- 뷰 홀더 준비
  - 각 항목에 해당하는 뷰 객체를 가지는 뷰 홀더는 RecyclerView.ViewHolder를 상속받아 작성

• 뷰 홀더 준비

```
class MyViewHolder(val binding: ItemMainBinding): RecyclerView.ViewHolder(binding.root)
```

- 어댑터 준비
  - 각 항목을 만들어 주는 역할

• 어댑터 준비

```
class MyAdapter(val binding: ItemMainBinding):
    RecyclerView.Adapter<RecyclerView.ViewHolder>() {
    override fun getItemCount(): Int {
        TODO("Not yet implemented")
    }
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
    RecyclerView.ViewHolder {
        TODO("Not yet implemented")
    }
    override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
        TODO("Not yet implemented")
    }
}
```

11-4 리사이클러 뷰 – 목록 화면 구성

- getItemCount(): 항목 개수를 판단하려고 자동으로 호출됩니다.
- onCreateViewHolder(): 항목의 뷰를 가지는 뷰 홀더를 준비하려고 자동으로 호출됩니다.
- onBindViewHolder(): 뷰 홀더의 뷰에 데이터를 출력하려고 자동으로 호출됩니다.

• 항목의 개수 구하기

```
override fun getItemCount(): Int = datas.size
```

• 항목 구성에 필요한 뷰 홀더 객체 준비

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder =
    MyViewHolder(ItemMainBinding.inflate(LayoutInflater.from(parent.context),
        parent, false))
```

• 뷰에 데이터 출력

```
override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
    Log.d("kkang","onBindViewHolder : $position")
    val binding = (holder as MyViewHolder).binding
    // 뷰에 데이터 출력
    binding.itemData.text = datas[position]
    // 뷰에 이벤트 추가
    binding.itemRoot.setOnClickListener {
        Log.d("kkang", "item root click : $position")
    }
}
```

11-4 리사이클러 뷰 – 목록 화면 구성

- 리사이클러 뷰 출력

• 리사이클러 뷰 출력

```
class RecyclerViewActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val binding = ActivityRecyclerViewBinding.inflate(layoutInflater)
        setContentView(binding.root)
        val datas = mutableListOf<String>()
        for(i in 1..10){
            datas.add("Item $i")
        }
        binding.recyclerView.layoutManager = LinearLayoutManager(this)
        binding.recyclerView.adapter = MyAdapter(datas)
        binding.recyclerView.addItemDecoration(DividerItemDecoration(this,
        LinearLayout Manager.VERTICAL))
    }
}
```

11-4 리사이클러 뷰 – 목록 화면 구성

- 항목을 동적으로 추가 · 제거
  - 항목을 구성하는 데이터에 새로운 데이터를 추가하거나 제거한 후 어댑터의 notifyDataSetChanged() 함수를 호출

• 항목 추가

```
datas.add("new data")
adapter.notifyDataSetChanged()
```

11-4 리사이클러 뷰 – 목록 화면 구성

레이아웃 매니저

- 레이아웃 매니저는 어댑터로 만든 항목을 리사이클러 뷰에 배치
  - LinearLayoutManager: 항목을 가로나 세로 방향으로 배치합니다.
  - GridLayoutManager: 항목을 그리드로 배치합니다.
  - StaggeredGridLayoutManager: 항목을 높이가 불규칙한 그리드로 배치합니다.

- 항목을 가로·세로 방향으로 배치
  - LinearLayoutManager를 사용

• 항목을 세로로 배치

```
binding.recyclerView.layoutManager =
    LinearLayoutManager(this)
```

▶ 실행 결과

11-4 리사이클러 뷰 – 목록 화면 구성

- LinearLayoutManager의 orientation값을 Linear LayoutManager.HORIZONTAL로 지정

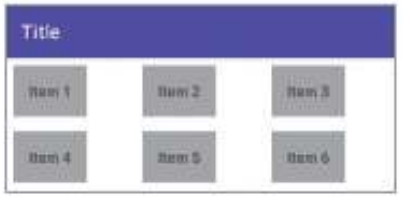
• 항목을 가로로 배치

```
val layoutManager = LinearLayoutManager(this)
layoutManager.orientation =
    LinearLayoutManager.HORIZONTAL
binding.recyclerView.layoutManager = layoutManager
```

▶ 실행 결과

11-4 리사이클러 뷰 – 목록 화면 구성

- 그리드로 배치하기
  - GridLayoutManager를 이용
  - 생성자의 숫자는 그리드에서 열의 개수를 뜻합니다
  - 방향을 설정할 수 있습니다.



- 가로로 설정하려면 생성자에 GridLayoutManager.HORIZONTAL을 지정

• 그리드에서 항목을 가로로 배치

```
val layoutManager = GridLayoutManager(this, 3,
GridLayoutManager.HORIZONTAL, false)
binding.recyclerView.layoutManager = layoutManager
```

▶ 실행 결과

11-4 리사이클러 뷰 – 목록 화면 구성

- GridLayoutManager 생성자의 네 번째 매개변수에 Boolean값을 설정

• 그리드에서 항목을 오른쪽부터 배치

```
val layoutManager = GridLayoutManager(this, 3,
GridLayoutManager.HORIZONTAL, true)
binding.recyclerView.layoutManager = layoutManager
```

▶ 실행 결과

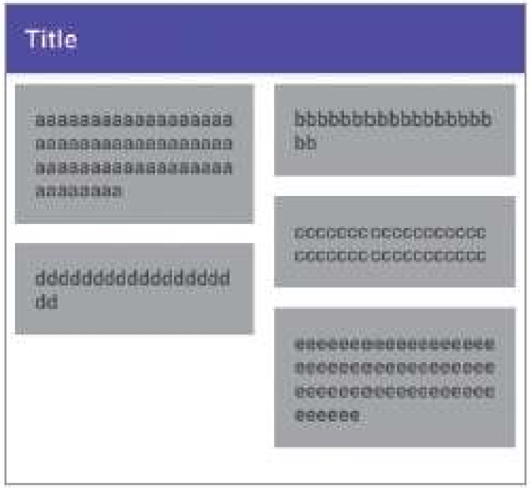
11-4 리사이클러 뷰 – 목록 화면 구성

- 높이가 불규칙한 그리드로 배치하기
  - StaggeredGridLayoutManager는 뷰의 크기가 다르면 지그재그 형태로 배치

• 지그재그 그리드 형태로 배치

```
val layoutManager = StaggeredGridLayout
Manager(2, StaggeredGridLayoutManager.
VERTICAL)
binding.recyclerView.layoutManager =
layoutManager
```

▶ 실행 결과



11-4 리사이클러 뷰 – 목록 화면 구성

아이템 데커레이션

- 아이템 데커레이션은 리사이클러 뷰를 다양하게 꾸밀 때 사용
- 항목의 구분선을 출력해 주는 DividerItem Decoration
- ItemDecoration을 상속받는 개발자 클래스를 만들고 이 클래스에서 다양한 꾸미기 작업을 합니다.
  - onDraw(): 항목이 배치되기 전에 호출됩니다.
  - onDrawOver(): 항목이 모두 배치된 후 호출됩니다.
  - getItemOffsets(): 개별 항목을 꾸밀 때 호출됩니다.

11-4 리사이클러 뷰 – 목록 화면 구성

• 아이템 데커레이션 구현

```
class MyDecoration(val context: Context): RecyclerView.ItemDecoration() {
    override fun onDraw(c: Canvas, parent: RecyclerView, state: RecyclerView.State) {
        super.onDraw(c, parent, state)
    }
    override fun onDrawOver(c: Canvas, parent: RecyclerView, state: RecyclerView.State) {
        super.onDrawOver(c, parent, state)
    }
    override fun getItemOffsets(
        outRect: Rect,
        view: View,
        parent: RecyclerView,
        state: RecyclerView.State
    ) {
        super.getItemOffsets(outRect, view, parent, state)
    }
}
```

11-4 리사이클러 뷰 – 목록 화면 구성

• 항목이 배치되기 전에 호출되는 onDraw() 함수

```
override fun onDraw(c: Canvas, parent: RecyclerView, state: RecyclerView.State) {
    super.onDraw(c, parent, state)
    c.drawBitmap(BitmapFactory.decodeResource(context.getResources(), R.drawable.stadium),
        0f, 0f, null)
```

11-4 리사이클러 뷰 – 목록 화면 구성

• 모든 항목이 배치된 후 호출되는 onDrawOver() 함수

```
override fun onDrawOver(c: Canvas, parent: RecyclerView, state: RecyclerView.State) {
    super.onDrawOver(c, parent, state)
    // 뷰 크기 계산
    val width = parent.width
    val height = parent.height
    // 이미지 크기 계산
    val dr: Drawable? = ResourcesCompat.getDrawable(context.getResources(),
        R.drawable.kbo, null)
    val drWidth = dr?.intrinsicWidth
    val drHeight = dr?.intrinsicHeight
    // 이미지가 그려질 위치 계산
    val left = width / 2 - drWidth?.div(2) as Int
    val top = height / 2 - drHeight?.div(2) as Int
    c.drawBitmap(
        BitmapFactory.decodeResource(context.getResources(), R.drawable.kbo),
        left.toFloat(),
        top.toFloat(),
        null
    )
}
```

11-4 리사이클러 뷰 – 목록 화면 구성

• 개별 항목을 꾸미는 getItemOffsets() 함수

```
override fun getItemOffsets(
    outRect: Rect,
    view: View,
    parent: RecyclerView,
    state: RecyclerView.State
) {
    super.getItemOffsets(outRect, view, parent, state)
    val index = parent.getChildAdapterPosition(view) + 1
    if (index % 3 == 0)
        outRect.set(10, 10, 10, 60) // left, top, right, bottom
    else
        outRect.set(10, 10, 10, 0)
    view.setBackgroundColor(Color.LTGRAY)
    ViewCompat.setElevation(view, 20.0f)
}
```

11-4 리사이클러 뷰 – 목록 화면 구성

- 아이템 데커레이션 객체를 리사이클러 뷰에 적용할 때는 addItemDecoration() 함수를 이용

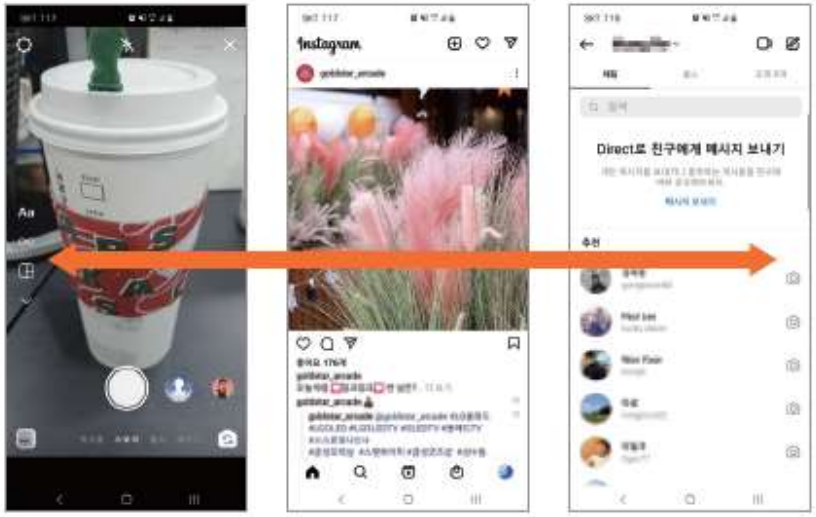
• 리사이클러 뷰에 아이템 데커레이션 적용

```
binding.recyclerView.addItemDecoration(MyDecoration
(this))
```

▶ 실행 결과

11-5 뷰 페이저2 – 스와이프로 넘기는 화면 구성

- 오랫동안 이용했던 viewpager와 별개로 2019년에 viewpager2를 제공.



• 뷰 페이저2 선언

```
implementation 'androidx.viewpager2:viewpager2:1.0.0'
```

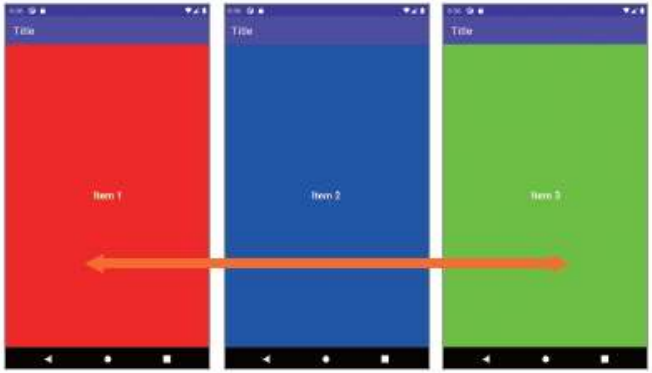
11-5 뷰 페이저2 – 스와이프로 넘기는 화면 구성

- 항목이 순서대로 나열되어 있는데 단지 한 화면에 항목 하나가 나온다는 개념
- 어댑터는 2가지인데 리사이클러 뷰에서 봤던 Recycler View.Adapter를 그대로 이용하거나 FragmentStateAdapter를 사용할 수도 있습니다.

- 리사이클러 뷰 어댑터 이용

• 뷰 페이저2 어댑터에 적용

```
binding.viewpager.adapter = MyPagerAdapter(datas)
```



• 뷰 페이저2 구현 - 리사이클러 뷰 어댑터 이용

```
class MyPagerViewHolder(val binding: ItemPagerBinding) :
    RecyclerView.ViewHolder(binding.root)
class MyPagerAdapter(val datas: MutableList<String>) :
    RecyclerView.Adapter<RecyclerView.ViewHolder>() {
    override fun getItemCount(): Int {
        return datas.size
    }
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
        RecyclerView.ViewHolder =
            MyPagerViewHolder(ItemPagerBinding.inflate(LayoutInflater.from(
                parent.context), parent, false))
    override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
        val binding = (holder as MyPagerViewHolder).binding
        // 뷰에 데이터 출력
        binding.itemPagerTextView.text = datas[position]
        when (position % 3) {
            0 -> binding.itemPagerTextView.setBackgroundColor(Color.RED)
            1 -> binding.itemPagerTextView.setBackgroundColor(Color.BLUE)
            2 -> binding.itemPagerTextView.setBackgroundColor(Color.GREEN)
        }
    }
}
```

11-5 뷰 페이저2 – 스와이프로 넘기는 화면 구성

- 프래그먼트 어댑터 이용
  - 프래그먼트로 작성했으면 FragmentStateAdapter로 뷰 페이저2를 구현

• 뷰 페이저2 구현 - 프래그먼트 어댑터 이용

```
class MyFragmentPagerAdapter(activity: FragmentActivity): FragmentStateAdapter(activity)
{
    val fragments: List<Fragment>
    init {
        fragments= listOf(OneFragment(), TwoFragment(), ThreeFragment())
        Log.d("kkang", "fragments size : ${fragments.size}")
    }
    override fun getItemCount(): Int = fragments.size
    override fun createFragment(position: Int): Fragment = fragments[position]
}
```

11-6 드로어 레이아웃 – 옆에서 열리는 화면 구성

- 레이아웃 XML 파일에서 드로어 메뉴가 출력되어야 하는 부분의 태그를 DrawerLayout으로 선언합니다.
- DrawerLayout 아래에는 뷰를 2개 선언
- 첫 번째 하위 태그 부분을 액티비티 화면에 출력
- 두 번째 하위 태그 부분이 안 보이다가 끌려 나옵니다.

• 드로어 레이아웃 구성

```
<androidx.drawerlayout.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout ... 생략 ... >
        (... 생략 ...)
    </LinearLayout>
    <TextView
        android:layout_gravity="start" />
</androidx.drawerlayout.widget.DrawerLayout>
```

11-6 드로어 레이아웃 – 옆에서 열리는 화면 구성

- 툴바 영역에 토글toggle 버튼을 함께 제공
- 토글 버튼은 ActionBarDrawerToggle 클래스에서 제공

• 드로어 메뉴 토글 버튼 적용

```
class DrawerActivity : AppCompatActivity() {
    lateinit var toggle: ActionBarDrawerToggle
    override fun onCreate(savedInstanceState: Bundle?) {
        (... 생략 ...)
        // ActionBarDrawerToggle 버튼 적용
        toggle = ActionBarDrawerToggle(this, binding.drawer, R.string.drawer_opened,
            R.string.drawer_closed)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        toggle.syncState()
    }
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        // 이벤트가 토글 버튼에서 발생하면
        if (toggle.onOptionsItemSelected(item)) {
            return true
        }
        return super.onOptionsItemSelected(item)
    }
}
```

제트팩을 이용한 화면구성하기

1단계. 새 모듈 생성하기

- Ch11_JetPack이라는 이름으로 새로운 모듈을 만듭니다.뷰

2단계. 빌드 그래들 작성하기

- 뷰 바인딩을 위한 설정

3단계. 문자열 리소스 만들기

- res/values/strings.xml 파일을 열어 ActionBarDrawerToggle을 생성할 때 지정할 문자열 리소스를 추가

Do it! 실습

시계 앱의 스톱워치 기능 만들기

4단계. 테마 변경하기

- res/values/themes.xml 파일을 열어 액티비티 윈도우를 출력할 때 액션바를 출력하지 않게 설정

5단계. 프래그먼트 생성하기

- [New → Fragment → Fragment (Blank)]를 선택
- OneFragment, TwoFragment, ThreeFragment라는 이름으로 빈 프래그먼트 3개를 추가합니다.

Do it! 실습 시계 앱의 스톱워치 기능 만들기

6단계. 파일 복사하기

- res/drawable/kbo.png 파일을 res/drawable에 복사
- es/layout 디렉터리의 fragment_one.xml, fragment_two.xml, fragment_three.xml, item_recyclerview.xml 파일을 res/layout 디렉터리에 복사
- OneFragment.kt, TwoFragment.kt, ThreeFragment.kt 파일을 소스 영역에 복사

7단계. fragment_one.xml 파일 작성하기

- 리사이클러 뷰로 구성하기 위해 fragment_one.xml 파일을 작성합니다.
- OneFragment.kt 파일을 작성

Do it! 실습 시계 앱의 스톱워치 기능 만들기

8단계. 메뉴 리소스 만들기

- [New → Android Resource Directory]를 선택합니다.
- Resource type 항목에서 [menu]를 선택하고 <OK>를 눌러 menu 디렉터리를 만듭니다.
- [New → Menu Resource File]을 선택합니다.
- File name에 menu_ main이라고 입력한 후 <OK>를 눌러 메뉴 파일을 만듭니다.
- menu_main.xml 파일을 열고 작성

9단계. 메인 레이아웃 XML 작성하기

- activity_main.xml 파일을 열고 드로어 레이아웃, 툴바, 뷰 페이저2 등을 등록합니다.

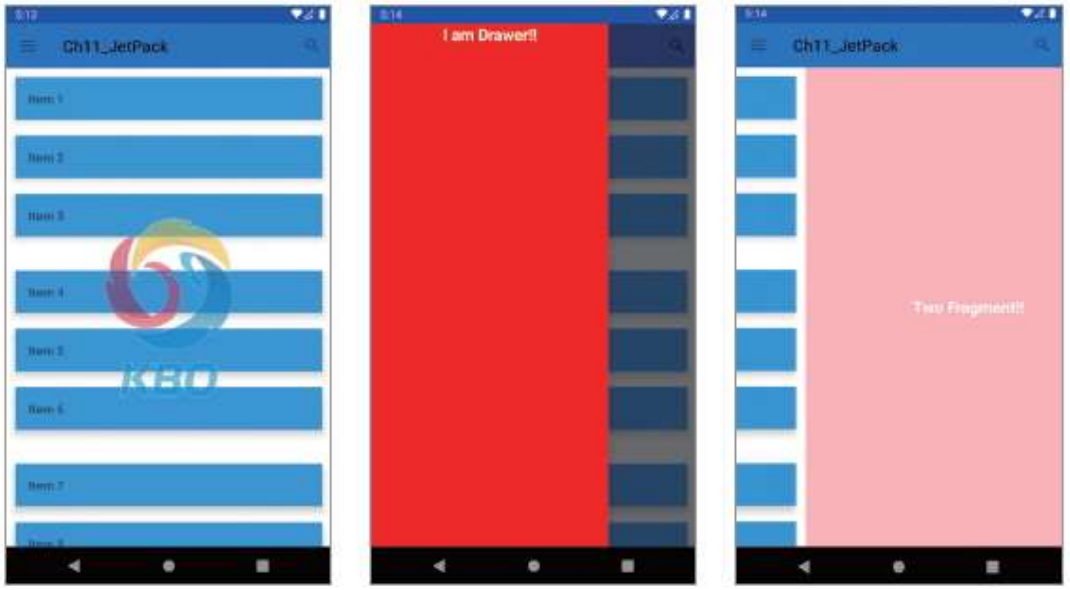
Do it! 실습

시계 앱의 스톱워치 기능 만들기

10단계. 메인 액티비티 작성하기

- MainActivity.kt 파일을 열고 작성

11단계. 앱 실행하기



감사합니다

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

윌리엄 셰익스피어
William Shakespeare