01-1. Fragment

androidx 뷰 활용

11-1 제트팩과 androidx 소개

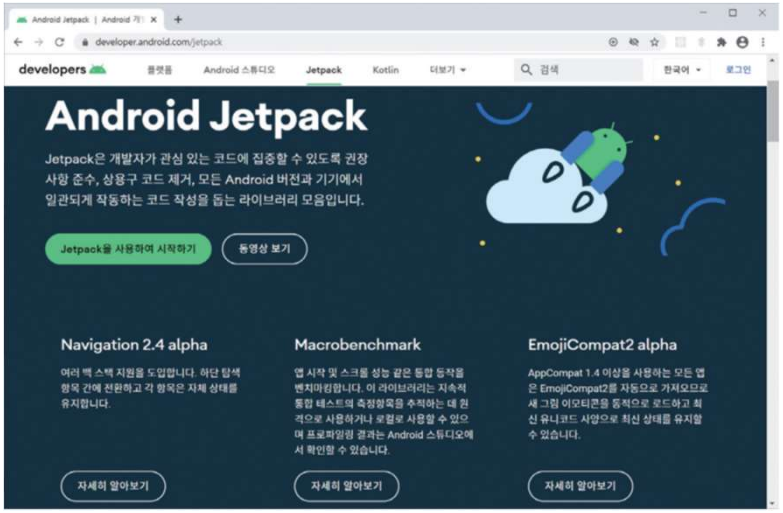
플랫폼 API

- 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(libs.androidx.appcompat)
```

• appcompat 라이브러리 사용

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

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 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
  - 프래그먼트를 출력하기 위해서는 FragmentContainerView가 제공

• 프래그먼트 출력

```
<androidx.fragment.app.FragmentContainerView xmlns:android="http://schemas.android.com/
apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/fragment_content"
    android:name="com.example.test_11.OneFragment">

</androidx.fragment.app.FragmentContainerView>
```

11-3 프래그먼트 – 액티비티처럼 동작하는 뷰

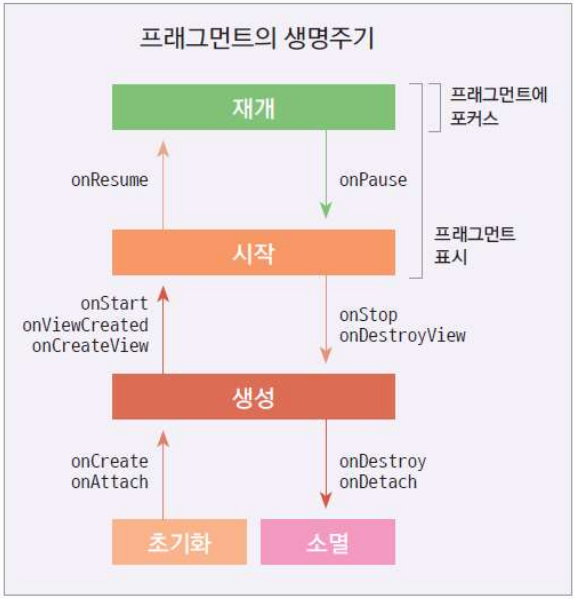
- 액티비티 코드에서 프래그먼트 출력
  - 프래그먼트를 동적으로 제어(추가, 제거 등)하려면 FragmentManager로 만든 FragmentTransaction 클래스의 함수를 이용
  - add(int containerViewId, Fragment fragment): 새로운 프래그먼트를 화면에 추가합니다.
  - replace(int containerViewId, Fragment fragment): 추가된 프래그먼트를 대체합니다.
  - remove(Fragment fragment): 추가된 프래그먼트를 제거합니다.
  - commit():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

감사합니다

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

윌리엄 셰익스피어
William Shakespeare