CHAPTER .3 안드로이드 베이직 by "Hello World"

3.1 프로젝트(Project) 생성과 실행

3.2 프로젝트의 구조

3.3 프로그래밍의 기본

3.4 컴파일(Compile)과 빌드(Build)

3.5 주석(Comment)의 이해

3.6 시작점(Entry Point)과 매니페스트(Manifest) 파일

3.7 변수와 데이터 형식

3.8 제어와 분기방법(if, else if, else, for, while)

3.9 함수와 메소드(Method), 리소스(Resource)

- method 는 객체 지향언어에서 특정 객체 안에 포함된 함수를 가리킴

- 안드로이드 스튜디오에서 리소스 분리를 통해 재사용성을 높일 수 있음

- 안드로이드에서 레이아웃은 리소스로 관리

activity\_main.xml 에서 include 항목

- 이미 다른 데에 정의되어 있는 레이아웃 리소스를 그대로 가져다 쓰는 것

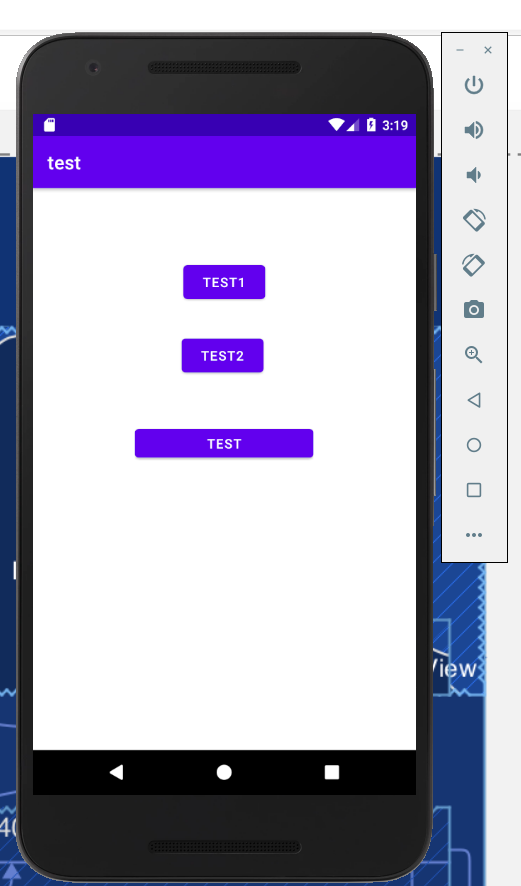
- 사용가능한 다양한 레이아웃 파일

infer constraints

- 레이아웃 내 배치된 위젯의 현재 상태를 기반으로 관계를 지정

- 이 기능은 화면에 포함되어 있는 전체 위젯을 대상으로 하여금 관계를 지정





3.10 객체 지향(OOP)과 클래스, 인스턴스

- 객체 지향 패러다임은 재사용성을 높이고 유지/보수를 쉽게 하지 위한 방법으로 '객체'를 제안

- 프로그램을 각각의 독립적인 객체 간의 상호작용으로 해석

- 객체 설계는 class 를 통해 정의, 생성된 객체는 instance라고 함

- class는 객체의 설계도, instance는 설계도를 통해 생성된 객체

- 캡슐화는 객체의 내부정보를 외부로부터 은닉하는 것

1. 내부의 복잡한 로직을 숨겨 외부에서 기능을 사용하기 쉽게 하기 위해

2. 내부의 주요속성들을 외부로부터 보호하는 것

- 자바에서 사용하는 접근 제어자는 4가지

1. public: 프로그램의 모든 위치에서 액세스 가능

2. protected: 동일한 패키지 또는 다른 패키지의 소브 클래스에서만 액세스 가능

3. default: 동일한 패키지 내에서만 액세스 가능

4. private: 자신의 클래스에서만 액세스가능. 클래스나 인터페이스는 private으로 지정할 수 없음

- 캡슐화 시, 필드는 private으로 하고 public 메소드를 통해 외부와 상호작용하는 것이 좋음

Car 프로젝트

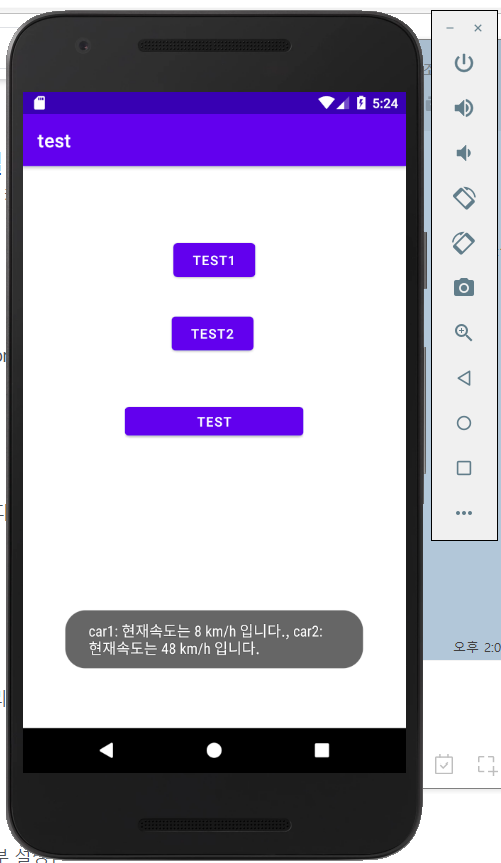
- Car 객체는 자신의 가속도와 최고속도에 따라 현재속도가 변하고, 현재의 정보를 화면에 메시지로 나타냄

- TEST2 버튼을 누르면 Car 객체의 생성자에서 지정한 브레이크 속도에 따라 감속

- 객체를 사용하여 코드의 공통되는 부분을 분리, 객체의 구체적인 작동은 은닉함으로써 재사용성이 높아짐







상속

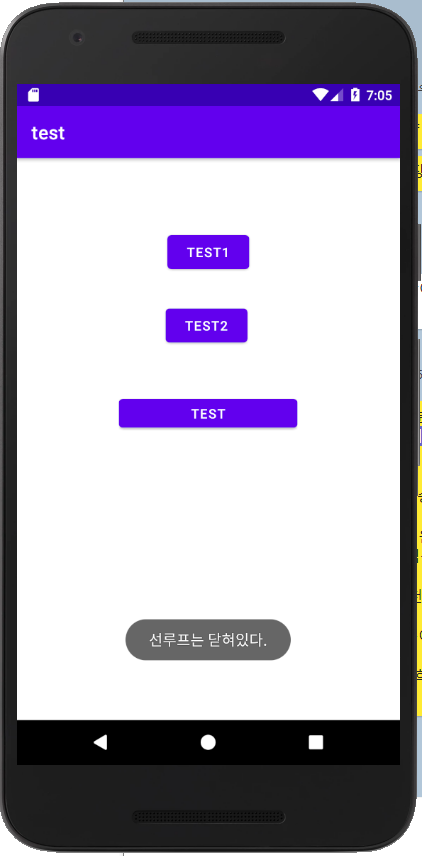
- 객체지향에서 코드를 재사용하기 위한 방법`

- 다른 클래스의 기능을 그대로 사용하면서 자신만의 기능을 추가하는 확장의 개념

- java에서는 상속/확장을 의미하는 extends 키워드를 사용







메소드 오버로딩

- 같은 기능을 하지만 입력 받는 '파라미터의 타입(자료형)'이 다를 때 사용

- 서로 다른 타입이어도 같은 메소드 이름을 쓸 수 있음

- 즉, 타입이 다르면 이름이 같아도 다른 덮어쓰지 않음

메소드 오버라이딩

- 상속에서 부모가 제공한 기능을 그대로 쓰지 않고 바꿔서 사용하는 것

- 자료형과 이름이 같지만 자식클래스만의 메소드를 만들어서 사용할 수 있다.

- 부모클래스를 자식클래스에서 직접적으로 사용하려면 super를 사용한다.

3.11 라이브러리, 프레임워크, 콜백

라이브러리

- 라이브러리란 개발의 생산성 향상을 위해 자주 사용하는 기능을 여러 곳에서 사용할 수 있도록 묶은 것

- test앱은 ConstraintLayout와 AppCompat 등의 라이브러리를 사용하고있음



프레임워크

- 프레임워크 역시 라이브러리처럼 자주 사용하는 기능을 묶어 개발 생산성 향상을 하는 것이 목적

- 라이브러리워 프레임워크의 차이점은 "코드 흐름의 제어권을 누가 가지고 있느냐"임

- 나의 메소드가 불려 사용되면 프레임워크라고 할 수 있음

콜백

- 프레임워크는 프로그램이 동작하는 일련의 과정 중 공통적인 요소들은 상위 클래스에서 정의,

- 서로 다르게 적용되어야 하는 부분은 하위 클래스에 구현된 'CallBack' 함수를 부르는 형태로 작동

- CallBack 함수는 '불려지는 함수'를 말한다.

- 이벤트 리스너의 'onClick' 함수는 버튼이 클릭될 때 불려오는 것 -> CallBack 함수

- OnCreate() 함수는 액티비티가 실행될 때 불려지는 CallBack 함수

% 테마, 스타일, 요소 등 공통적인 부분은 모든 앱이 유사하게 때문에 프레임워크에서 처리하고,

서로 다른 부분은 CallBack 함수를 불러 사용한다.

4. Kotlin(코틀린) 실전 프로그래밍

코틀린의 특징

- 코틀린은 자바와 100% 상호호환 되면서도 더 현대적이고 간결. 한 개의 프로젝트 내에서 혼용가능

- null pointer exception 발생 코드 금지하고 null 에 대해 안전한 코드를 지원

- 타입추론을 지원하여 동적타이핑 가능

- 객체지향과 함수형 프로그래밍 둘 다 지원

- 코틀린의 확장함수는 기존 클래스 변경 없이 클래스 기능 추가하는 것이 가능

- 공식언어이기 때문에 안드로이드에서 확장기능 제공. 반복코드 제거 가능

코틀린과 자바의 차이점

- 함수를 선언할 때 ‘fun’이라는 키워드를 사용한ㄷ

- 파라미터 이름 뒤에 파라미터 타입을 쓴다. 자바는 String[] args로 타입을 먼저쓰고 이름을 쓰지만, 코틀리은 args: Array<String> 형식을 쓰고 있다.

- 세미콜론(;)을 쓰지 않는다.

- 함수의 반환값이 없는 경우 void를 쓰지 않고 그냥 생략한다.

4.3 확장 플러그인

- 확장 플러그인은 특정 환경을 위해 부가적으로 제공하는 기능을 말한다.

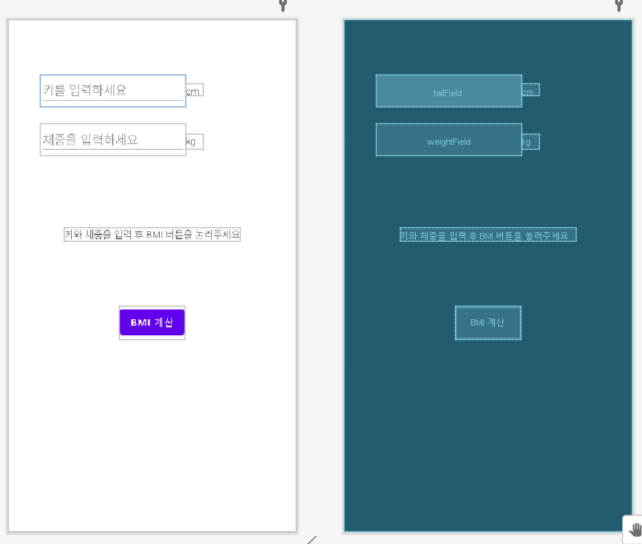
- 코틀린 자체는 안드로이드만을 위한 언어가 아님. 그래서 확장 플러그인으로 안드로이드를 위한 기능을 추가로 제공

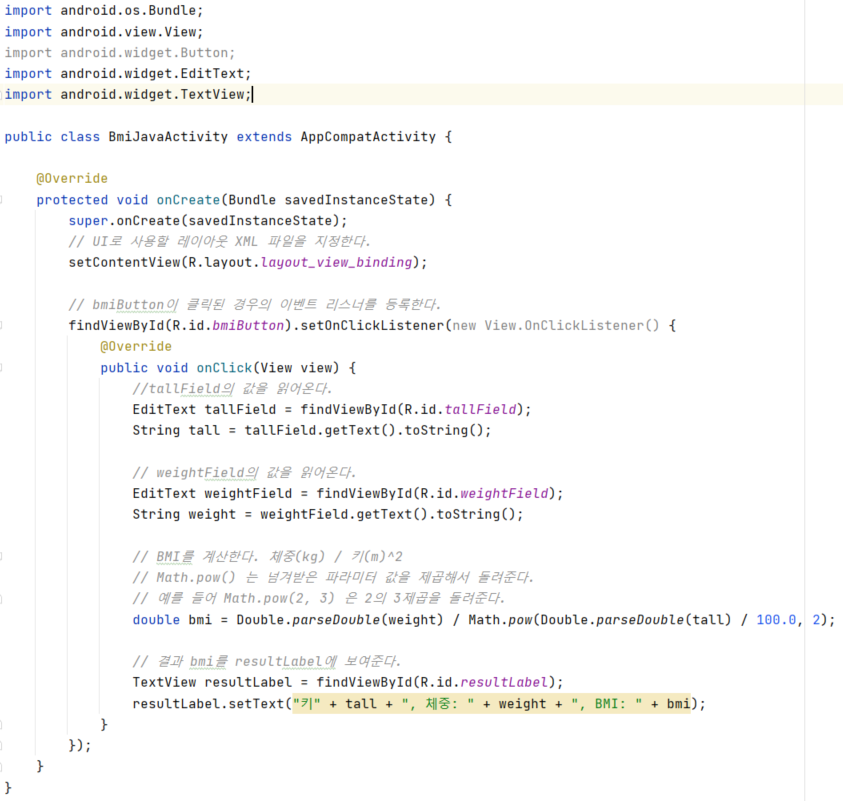
- bulid.gradle 파일에 플러그인 추가 “id ‘kotlin-android-extensions’”

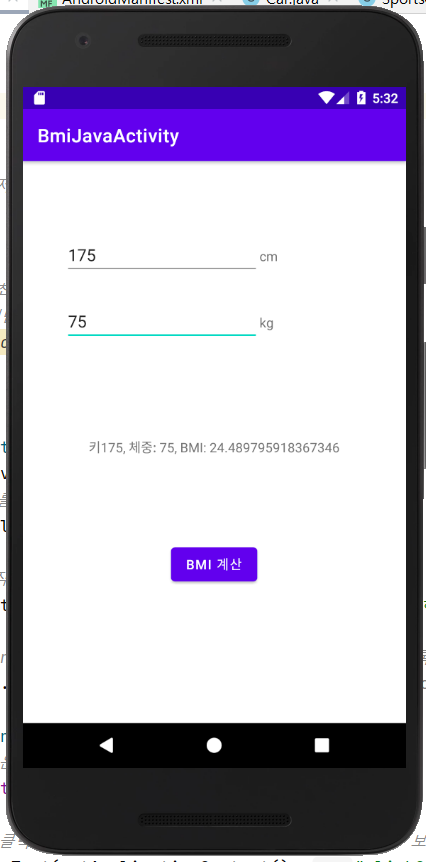
1. findViewById() 없이 접근하는 View Binding 기능

2. 객체를 직렬화할 때 사용하는 parcelable을 위한 기능

- 키와 몸무게를 입력받아 자바와 코틀린으로 실행되는 어플 만들기







- 하지만 이 방법은 버튼을 클릭할 때마다 findViewById() 함수를 실행하기 때문에 성능이 매우 안좋다. 그래서 findViewById()는 최초 실행 시 호출되고 변수에 담는 것이 일반적이다. 이 방법으로 작성하기

- 코틀린은 view binding 작업이 없어서 코드가 훨씬 간결해질 수 있다.

- 코틀린의 안드로이드 확장 플러그인은 view binding, parcelable 어노테이션을 지원한다.

- view binding 기능은 xml로 작성된 ui 요소를 findViewById() 함수없이 직접 접근이 가능하도록 한다.

4.4 변수

- 코틀린은 변수를 선언할 때 ‘var’, ‘val’ 키워드를 사용

- 변수의 불변성과 타입추론을 위해 두가지의 키워드를 사용한다.

4.4.7 변수의 가변성과 불변성

- 처음으로 내부속성을 설정한 이후로 변경이 불가능한 객체를 ‘불변객체’라고 함

- 자바에서 변수의 불변성을 표현하기 위해 ‘변수 앞에 final’을 붙인다.

- Activity의 시작시간은 처음에 값을 한번 설정하면 더 이상 값을 변경할 필요가 없음

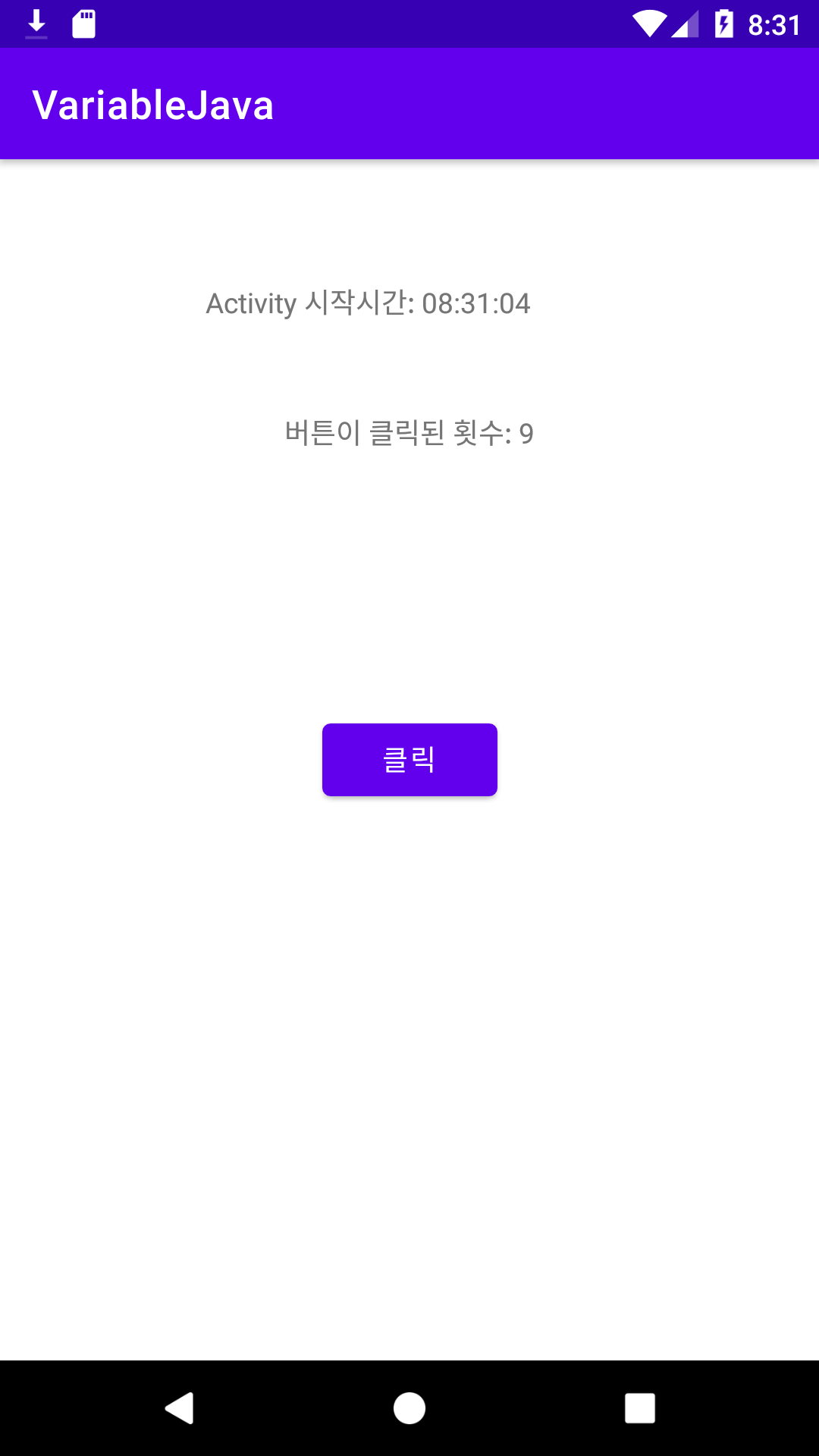
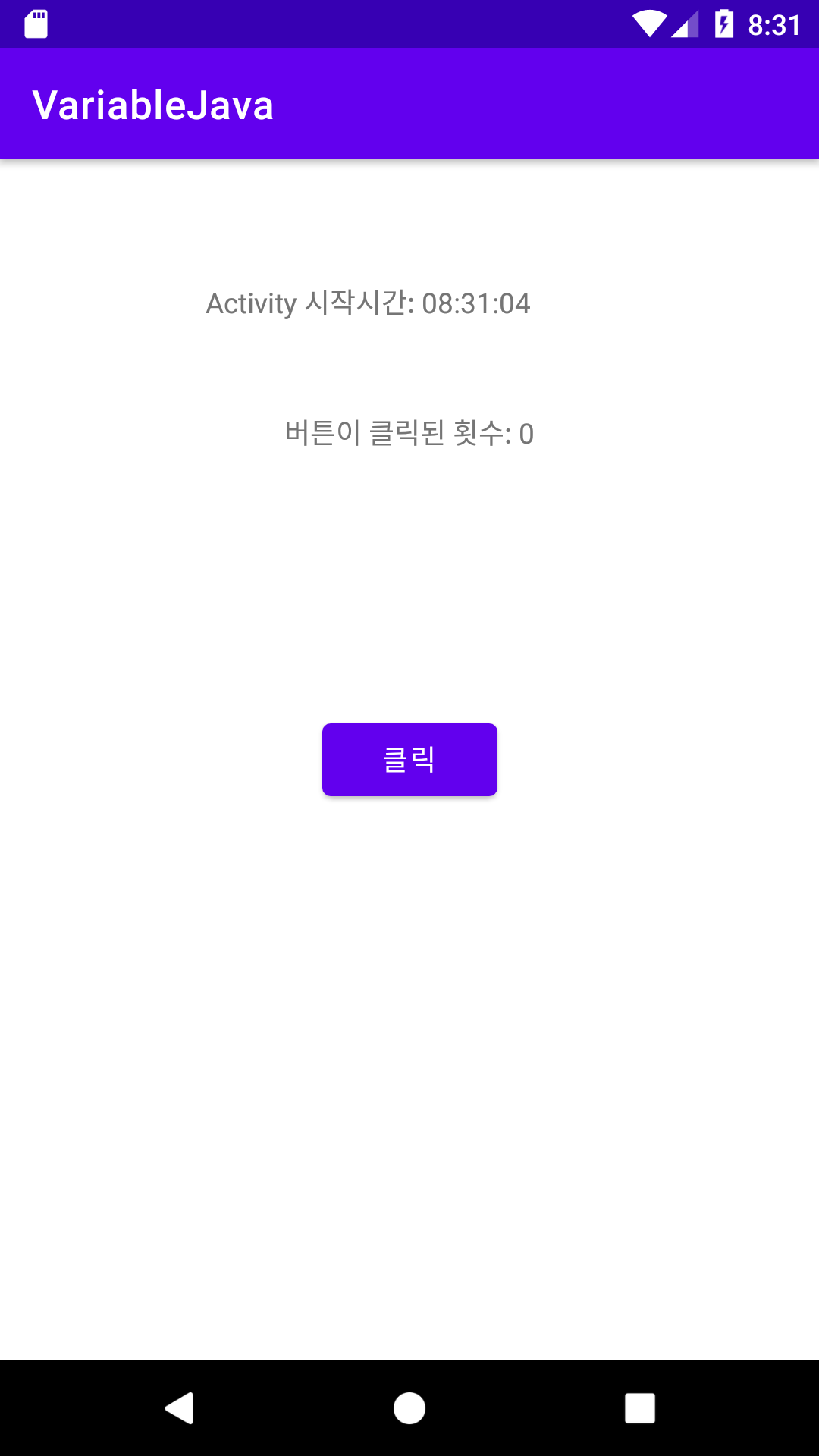
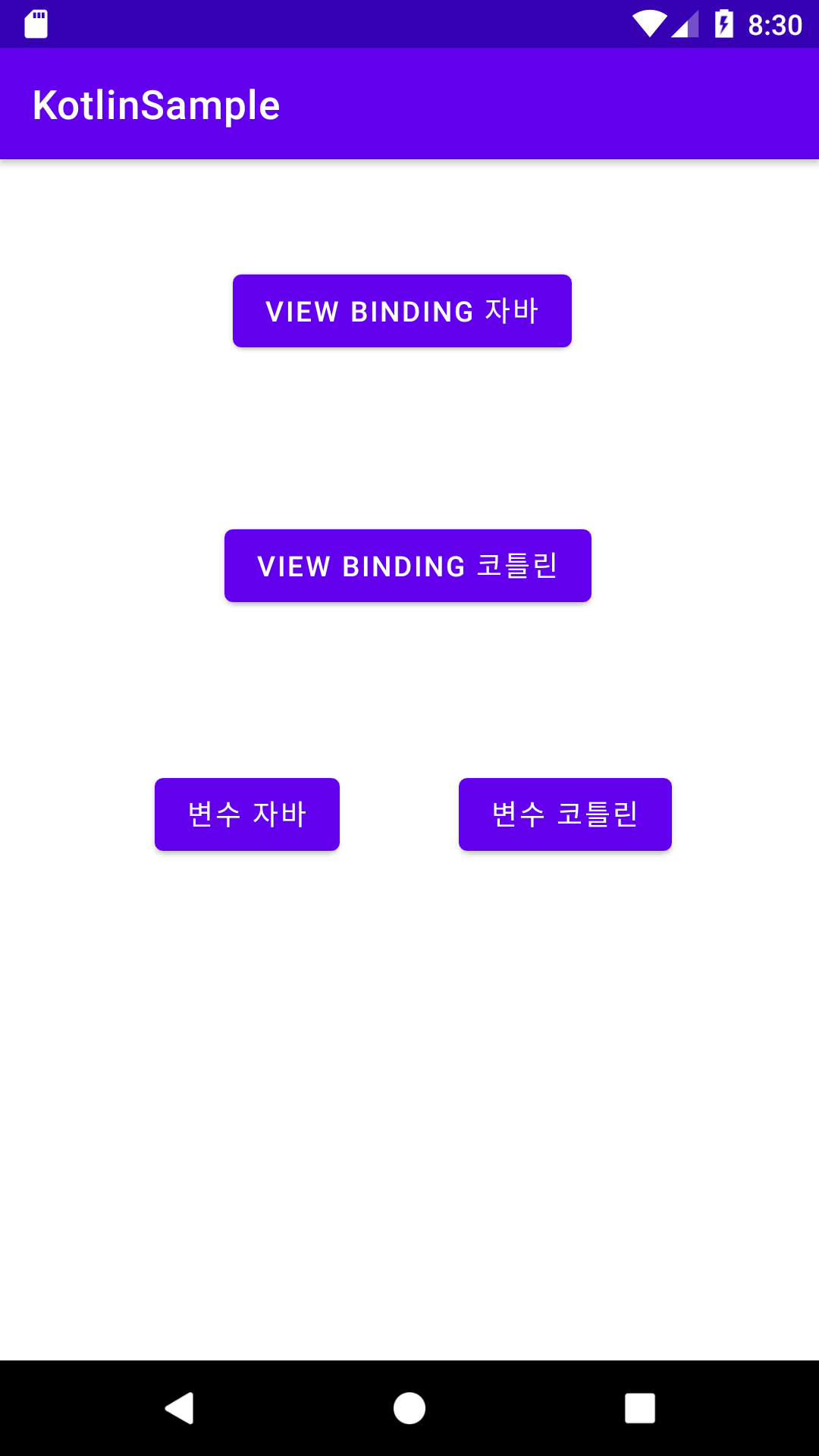
- 불변객체로 관리하면 메모리도 더 효율적으로 관리할 수 있음

- 멀티쓰레딩 할때도 효과적, 서로 불변객체를 건드리지 못하기 때문에

- 함수형 코드를 짤 때 유용하다. 함수형 코드란 같은 입력을 받으면 항상 같은 결과가 나오도록 코딩

- 기능을 알아보기 쉽게 함수로 구현해놓고 객체에서 사용하는 것

[액티비티 시작시간 보여주고 버튼 클릭할 때마다 클릭횟수 증가시켜 화면에 보여주는 어플]



- 불변변수는 ‘val’ (value), 가변변수는 ‘var’ (variable)

- 코틀린은 가급적 모든 변수를 ‘val’로 표현하고 필요한 경우에만 ‘var’ 를 쓰도록 권장함

- 코틀린은 변수의 타입을 적을 수도 있지만 변수의 타입을 굳이 적지 않는 것은 타입추론때문

4.4.9 변수의 타입추론

var clickCount = 0

* 코틀린은 정적타이핑 언어이기 때문에 타입이 결정되어 있음. 그래서 실행중에 타입을 바꿀 수 없다.
* 코틀린에서 타입을 생략하지 않고 쓰면 다음과 같음

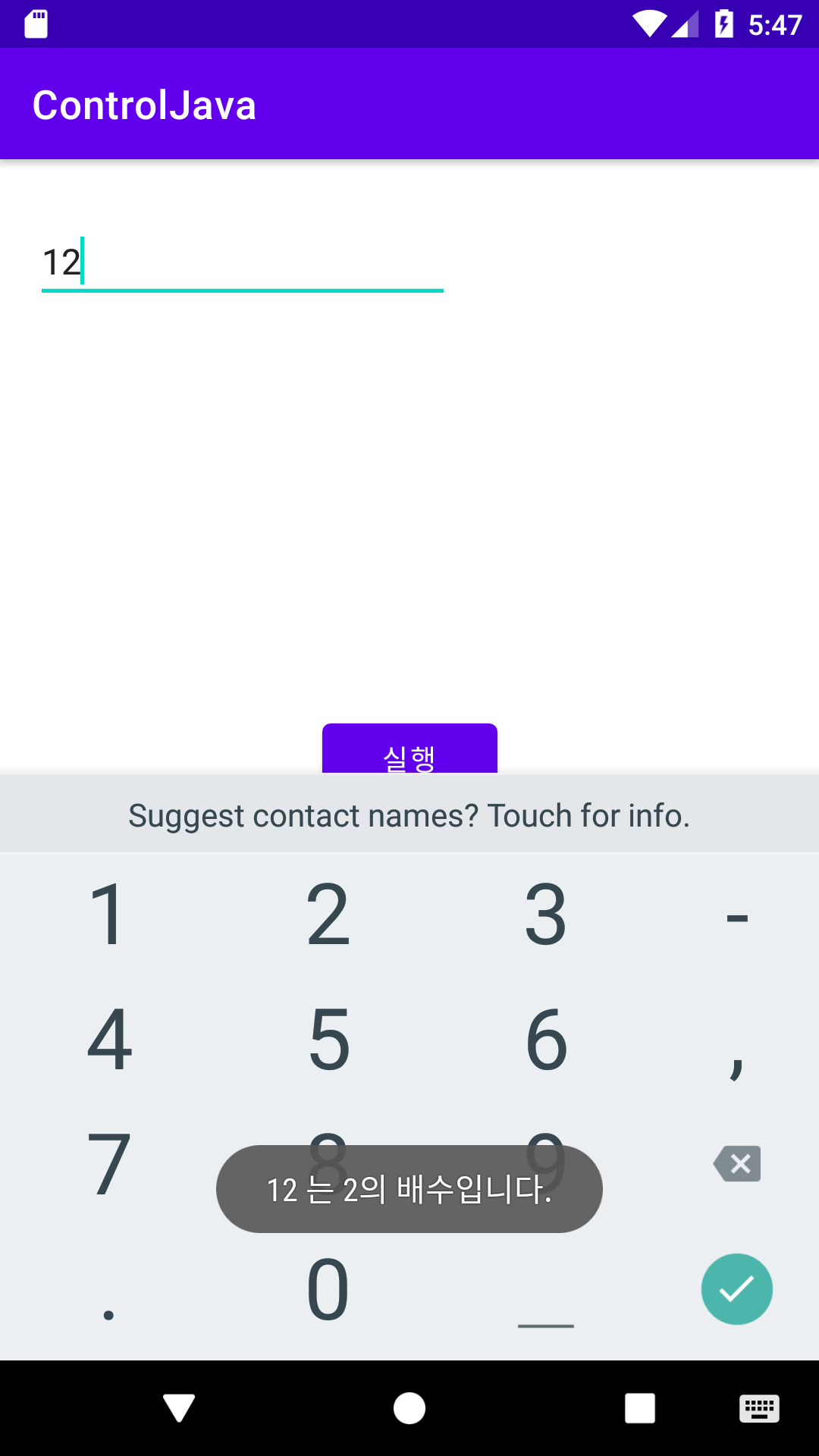
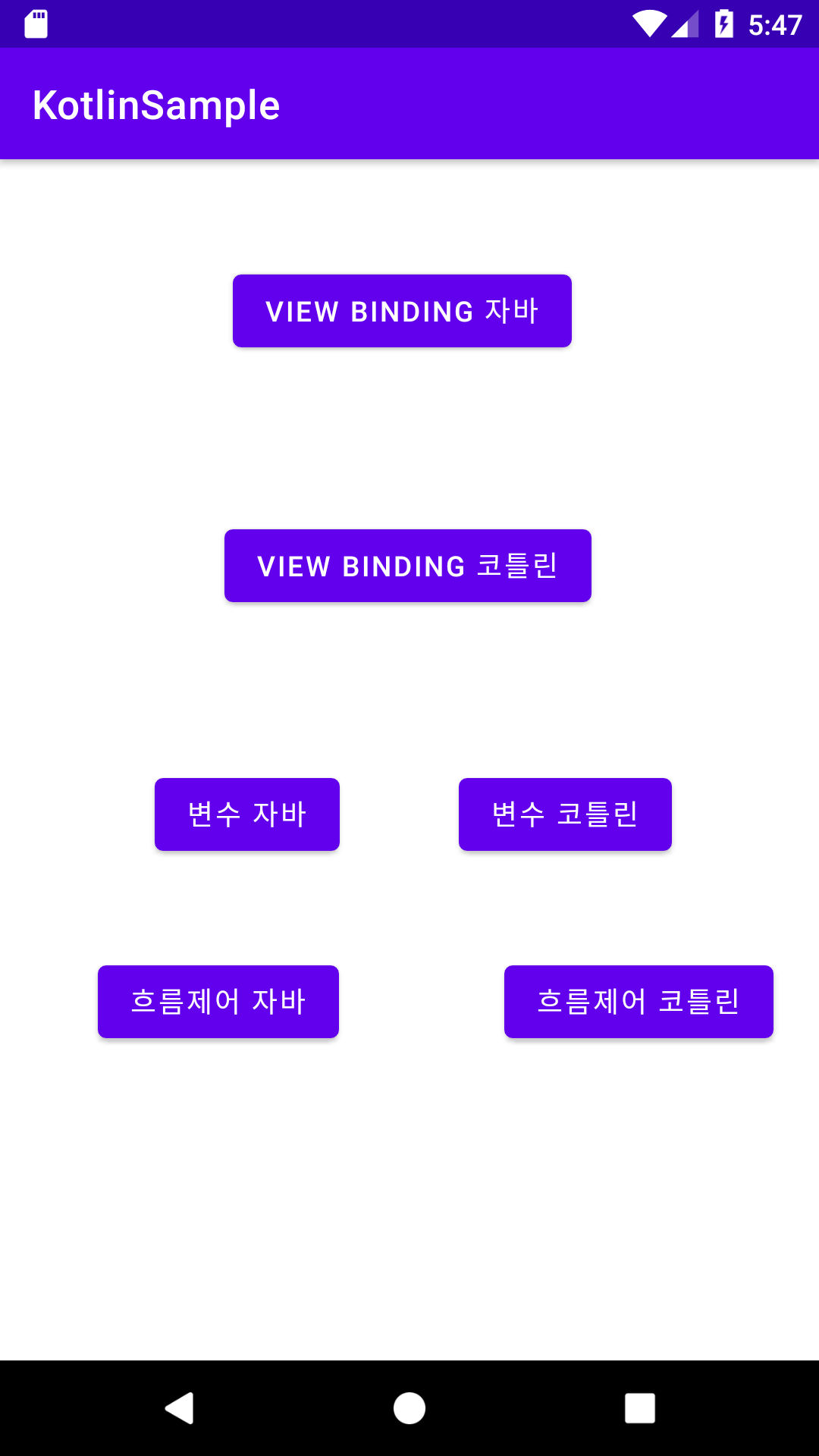
var clickCount: Int = 0

* 변수의 타입을 이름 뒤에 사용
* 코틀린에서 타입을 지정하지 않기 때문에 동적타입언어라고 생각할 수 있으나 정적타입언어이다.!!
* 코틀린은 변수의 타입을 체크한 경우 블록 내에서 타입 캐스팅이 자동으로 되는데 이것을 ‘스마트 캐스팅이라고 한다.

4.5 프로그램 흐름 제어

- 코틀린의 when 구문은 break를 넣지 않아도 한 개의 case만 실행된다.

[입력하는 숫자에 따라서 textview와 toast로 출력되는 값이 달라지는 어플]



* 코틀린의 조건문은 java와 같은 방법으로 if ~ else문을 사용할 수 있다.
* when문은 switch 문과 다중 if ~ else 문을 대체할 수있으며 훨씬 더 간결하다
* 코틀린은 반복문으로 for, whilem do ~ while 문을 사용할 수 있다.
* 코틀린은 반복문에 레이블을 지정할 수 잇으며 break 문 사용 시 해당 레이블로 빠져나갈 수 있다.

loop1@ while (true) {

……

break@loop1

이런식으로~

4.6 함수

자바

public void function(int age) { }

public int function(int age) { return 0; }

코틀린

fun function(age:int) { } // 반환값이 없으면 굳이 적지 않는다

fun function(age:int): int { return 0 } // 반환값은 함수 선언부 끝에 :을 사용해서

최상위함수 - 코틀린을 쓴다면 클래스를 만들지 않고 전역함수를 만들 수 있다.

최상위함수는 유틸리티 함수를 선언할 때 매우 유리하다.

사실 코틀린의 최상위함수는 함수가 없는 상태가 아니다!! 최상위함수는 컴파일 시점에 자동으로 새로운 클래스에 포함된다. (java와의 호환 때문에)

최상위함수가 위치한 코틀린 파일의 이름을 보고 자동으로 클래스를 만든다.

ToastUtilKotlin -> ToastUtilKotlinKT 클래스

즉 자바에서 코틀린의 최상위 함수를 쓸 때는 클래스 이름을 유추해서 사용하면 된다.

자동으로 생성되는 클래스이름 대신 다른 이름을 사용하려면 @file 어노테이션을 활용가능

@file:JvmName(“ToastUtil”)