App: WordQuiz

Mobile Software

2022 Fall

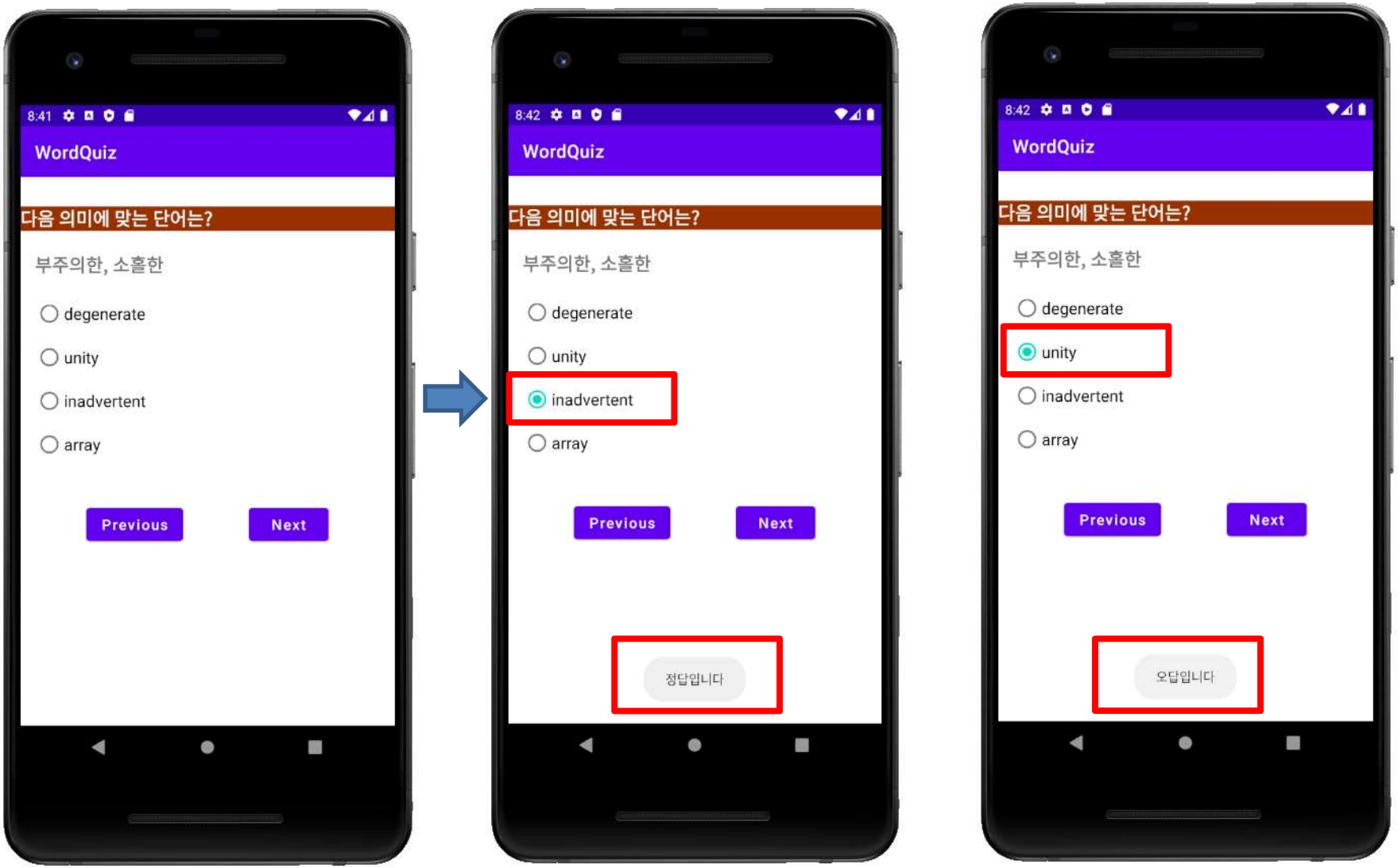
실습 목적

- **TOEIC 단어 공부를 위한 App. 구현**
  - **단계별 App 기능 개선**
    - 기본 App. → MVC 구조로 변환 → 화면 전환
      - 레이아웃 설명은 최대한 간단히 → 소스코드 참조(**wordquiz.hwp**)
  - **화면 구성 이해**
    - Layout: ConstraintLayout(**chain**), TableLayout, …
      - **tools:android** 속성, strings.xml, 이미지, …
  - **Kotlin 코드 이해**
    - build.gradle에 외부 라이브러리 추가
    - data class, ViewModel, companion object, …
    - 이벤트 처리
    - Intent를 사용한 화면 전환

단계 1: TOEIC 단어 퀴즈

- **단계 1: Basic App**
  - **화면 구성(ConstraintLayout)**
    - **문자열 리소스**
  - **문제 은행(data class)**
  - **문제 이동(Next, Previous) 기능**
  - **정답 확인**
- **단계 2: MVC 구조로 변경**
  - MVC – 문제 은행(ViewModel)
- **단계 3: 새 화면 추가**
  - 화면 구성(LinearLayout + TableLayout)
  - 단어 사전
  - Hint 제공: 2개의 Activity

단계 1: Basic App



단계 1: Project 구성

새 프로젝트 생성

Project name : **WordQuiz**

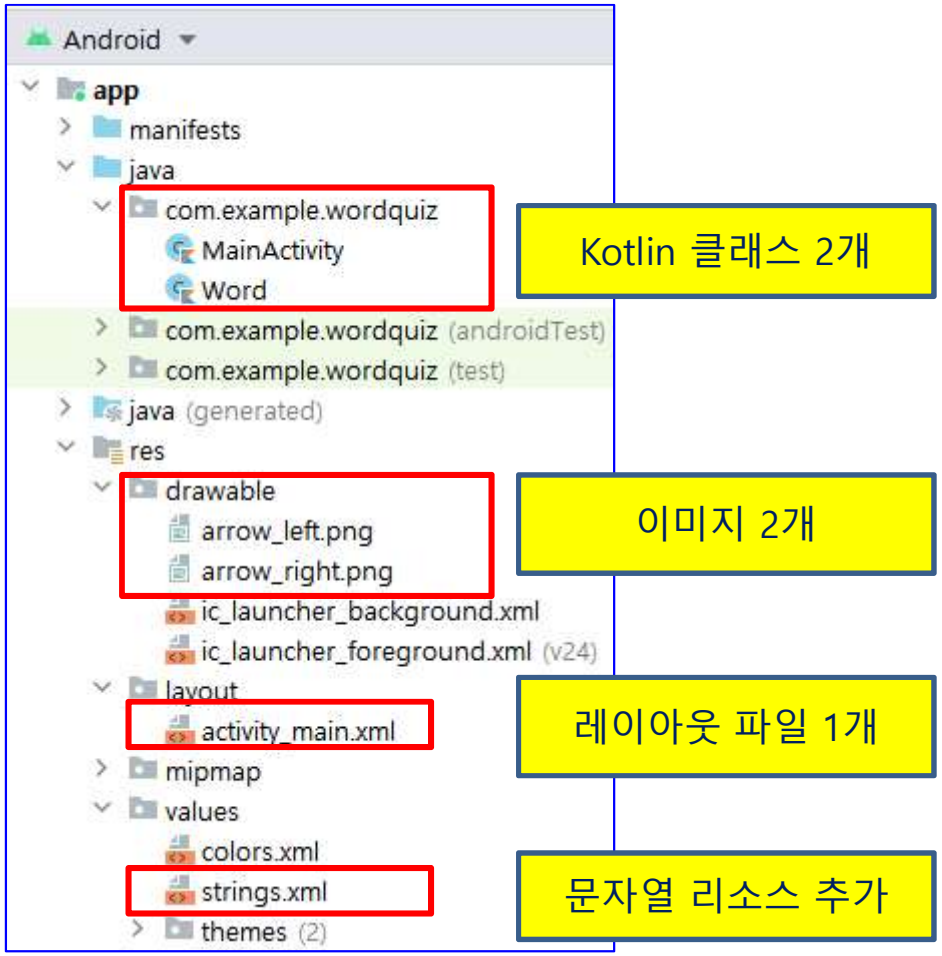
Package name :

com.example.wordquiz

Activity : **Empty Activity**

Activity name : **MainActivity.kt**

Layout name : **activity_main.xml**

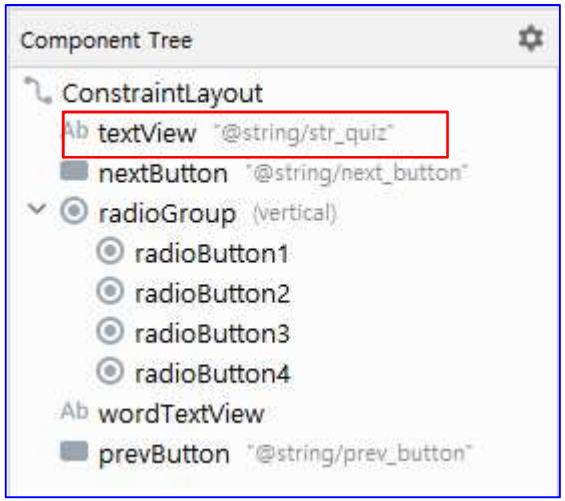


문자열 리소스 - strings.xml

상수 문자열을 코드에서 직접 사용하기 보다는
문자열 리소스로 정의하고 관리하는 게 바람직함

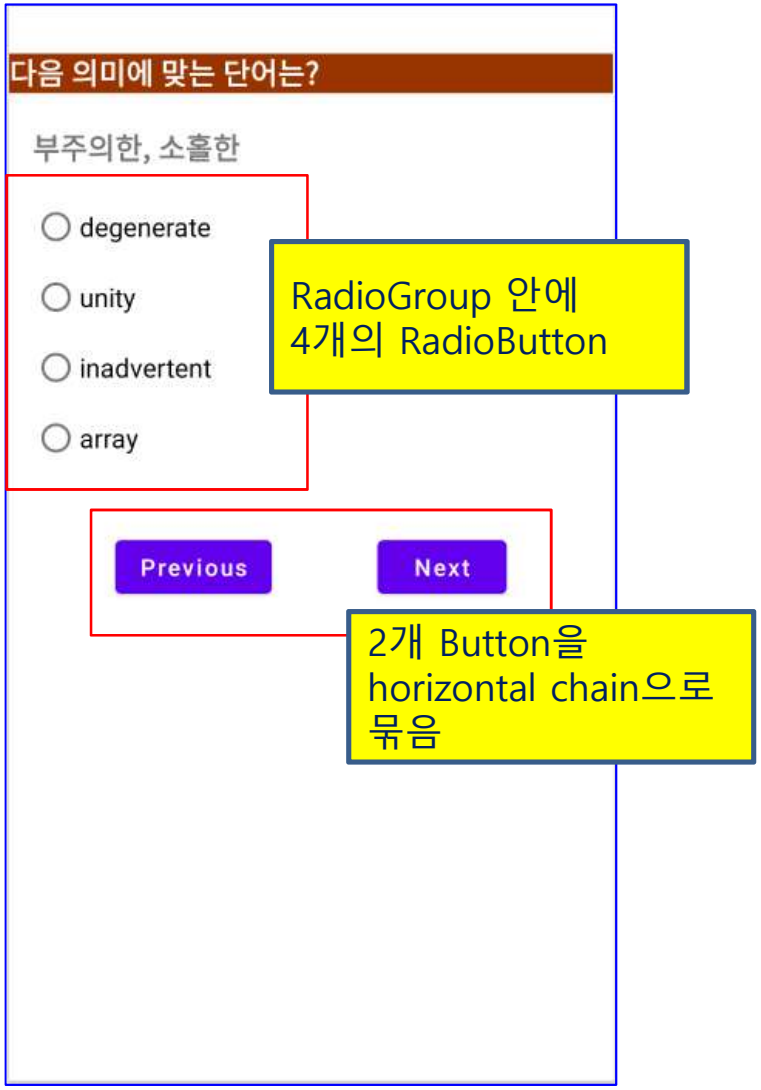
```
<resources>
    <string name="app_name">WordQuiz</string>
    <string name="str_quiz">다음 의미에 맞는 단어는?</string>
    <string name="next_button">Next</string>
    <string name="prev_button">Previous</string>
</resources>
```

Layout – activity_main.xml

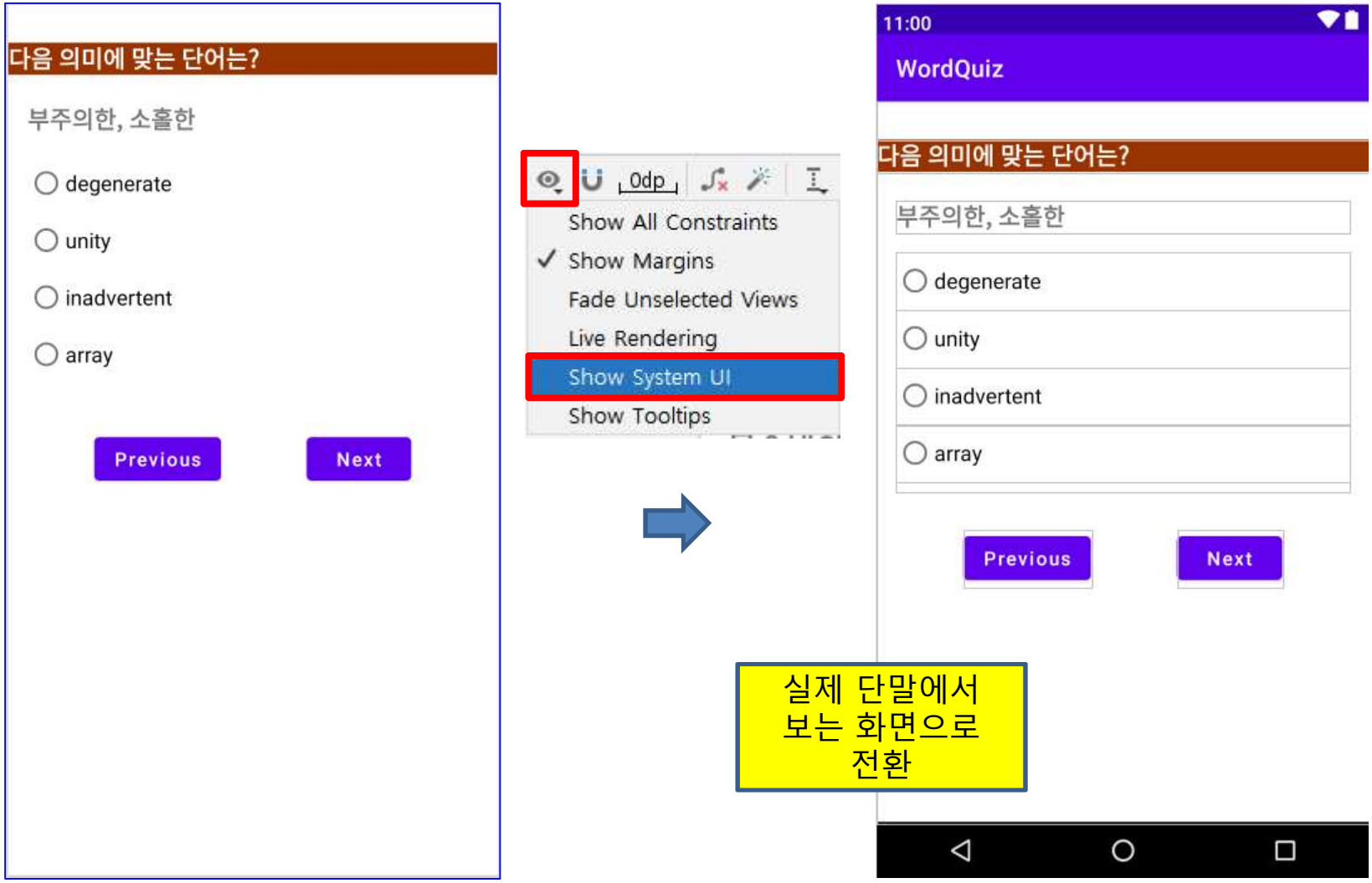



```
<RadioButton
    android:id="@+id/radioButton1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    tools:text="degenerate" />
```

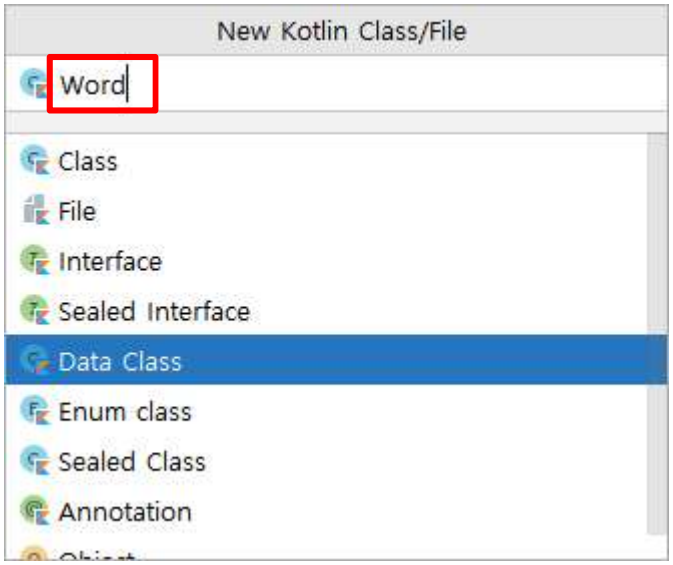
레이아웃 설계할 때만 Preview에서 볼 수 있음. AVD나 실제 단말에는 나타나지 않음.



잠깐! Preview 화면 전환



data class - Word




```
package com.example.wordquiz

data class Word(val question: String,
                val number_1: String,
                val number_2: String,
                val number_3: String,
                val number_4: String,
                val answer:Int )
```

데이터 클래스는 구현이 목적이 아니라
데이터 저장을 위한
틀(schema)을 제공.

data class Word (...)
클래스 이름 뒤의 괄호는
기본 생성자(primary constructor)

기본 생성자 에서
6개의 property를 선언
(문제, 4개의 지문, 정답)

화면에 문제 출력 – Activity (1/2)

클래스 이름과 멤버 메소드 사이에 선언한 변수는 클래스의 속성
➔ **클래스 안에서 자유롭게 참조할 수 있음.**

List 컬렉션에 Word 객체를 **순서대로 추가.**

```
class MainActivity : AppCompatActivity() {

    private lateinit var wordTextView: TextView
    private lateinit var radioButton1: RadioButton
    private lateinit var radioButton2: RadioButton
    private lateinit var radioButton3: RadioButton
    private lateinit var radioButton4: RadioButton

    private val wordBank = listOf<Word>(
        Word("부주의한, 소홀한", "degenerate", "unity", "inadvertent", "array", 3),
        Word("쇠약하게 하다", "vanity", "underhand", "enslave", "debilitate", 4),
        Word("(위험·곤란)제거하다", "artisan", "sadistic", "glossy", "obviate", 4),
        Word("우아한", "prostrate", "delude", "urbane", "renowned", 3),
        Word("활기있게 하다", "bereave", "enliven", "occult", "besiege", 2)
    )
    private var curIndex = 0

    override fun onCreate(savedInstanceState: Bundle?) {...}
}
```

클래스 속성은 선언과 함께 초기화가 필요. 그러나 view 객체들은 **setContentView** 이후에 참조가 가능하기 때문에 지금 초기화할 수 없음.
➔ 키워드 **lateinit** 가 앞에 붙은 속성들은 나중에 초기화하겠다는 의미.
(initialize the value **late)**

화면에 문제 출력 – Activity (2/2)

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

    wordTextView = findViewById(R.id.wordTextView)
    radioButton1 = findViewById(R.id.radioButton1)
    radioButton2 = findViewById(R.id.radioButton2)
    radioButton3 = findViewById(R.id.radioButton3)
    radioButton4 = findViewById(R.id.radioButton4)

    wordTextView.text = wordBank[curIndex].question
    radioButton1.text = wordBank[curIndex].number_1
    radioButton2.text = wordBank[curIndex].number_2
    radioButton3.text = wordBank[curIndex].number_3
    radioButton4.text = wordBank[curIndex].number_4
}
```

RadioButton과 TextView (문제)에 해당하는 데이터 연결

Next 버튼과 Previous 버튼 이벤트 처리 및 정답 처리는 하지 않은 상태

문제 이동 – Activity

Next 버튼 이벤트 처리
→ 다음 문제로 이동

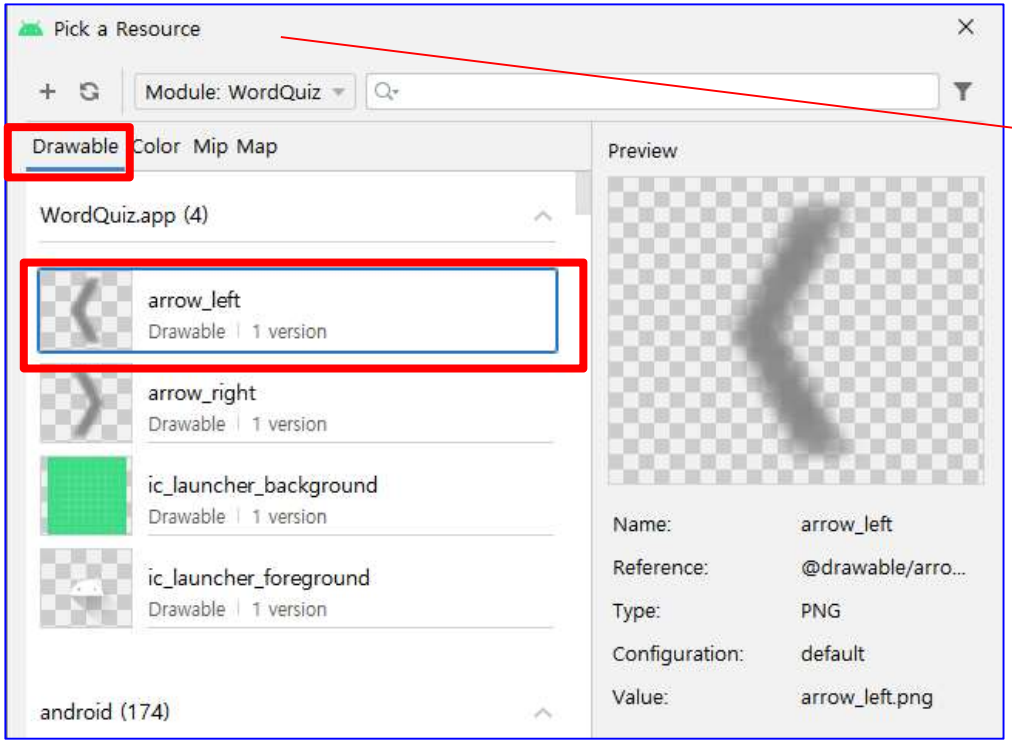
Previous 버튼 이벤트 처리
→ 이전 문제로 이동

해당 문제를
화면에 출력

```
override fun onCreate(savedInstanceState: Bundle?) {

    nextButton = findViewById(R.id.nextButton)
    prevButton = findViewById(R.id.prevButton)

    updateWordQuiz()

    nextButton.setOnClickListener {  it: View!
        curIndex = (curIndex + 1) % wordBank.size
        updateWordQuiz()
    }
    prevButton.setOnClickListener {  it: View!
        if (curIndex == 0) {
            curIndex = wordBank.size - 1
        } else {
            curIndex -= 1
        }
        updateWordQuiz()
    }
}

private fun updateWordQuiz() {
    wordTextView.text = wordBank[curIndex].question
    radioButton1.text = wordBank[curIndex].number_1
    radioButton2.text = wordBank[curIndex].number_2
    radioButton3.text = wordBank[curIndex].number_3
    radioButton4.text = wordBank[curIndex].number_4
}
```

문제를 바꾸는 것
➔ curIndex 속성 값 변경

잠깐! 버튼에 이미지 추가




```
<Button
    android:id="@+id/prevButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:drawableLeft="@drawable/arrow_left"
```

문제 채점 - Activity

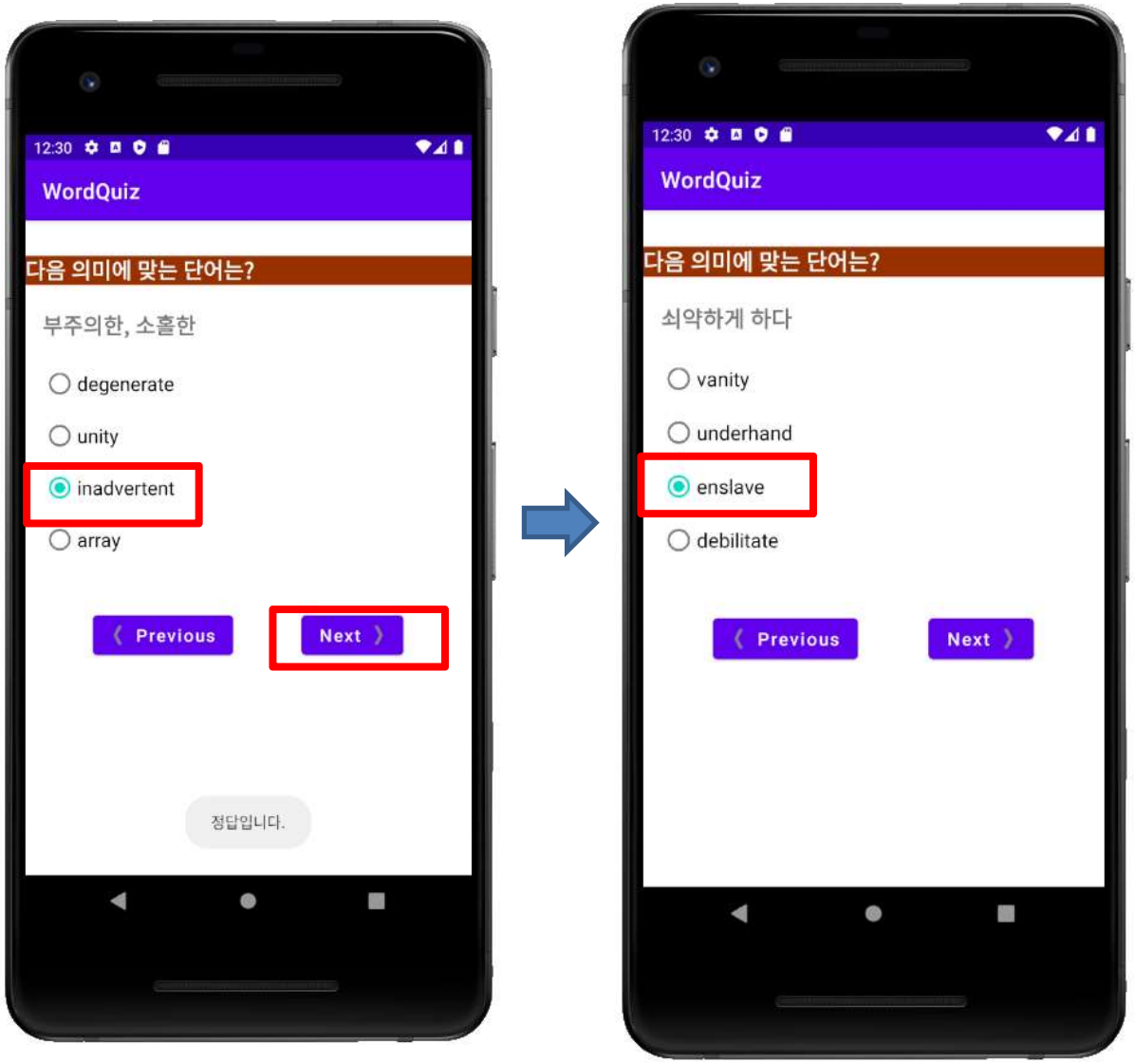
```
override fun onCreate(savedInstanceState: Bundle?) {
    nextButton.setOnClickListener {...}

    prevButton.setOnClickListener {...}

    radioButton1.setOnClickListener {  it: View!
        checkAnswer(1)
    }
    radioButton2.setOnClickListener {  it: View!
        checkAnswer(2)
    }
    radioButton3.setOnClickListener {  it: View!
        checkAnswer(3)
    }
    radioButton4.setOnClickListener {  it: View!
        checkAnswer(4)
    }
}
```

```
private fun checkAnswer(userAns: Int) {
    val correctAns = wordBank[curIndex].answer

    val message: String = if (userAns == correctAns) {
        "정답입니다"
    } else {
        "오답입니다"
    }
    Toast.makeText(this, message, Toast.LENGTH_SHORT)
        .show()
}
```

Quiz #1

BUG ALERT!
이전 문제에서 선택했던
RadioButton 상태가
다음 문제로
이동했을 때
그대로 남아있습니다.

**BUG가
없어지도록 코드를
수정하세요!**



Quiz #2

상수 문자열을 코드에서
직접 사용하기 보다는
strings.xml에
정의하고
관리하는 게 좋습니다.

상수 문자열 리소스를
참조하도록
코드를 수정하세요!

```xml
<resources>
    <string name="app_name">WordQuiz</string>
    <string name="str_quiz">다음 의미에 맞는 단어는?</string>
    <string name="next_button">Next</string>
    <string name="prev_button">Previous</string>
    <string name="right_ans_msg">정답입니다.</string>
    <string name="wrong_ans_msg">오답입니다.</string>
</resources>
```

```
private fun checkAnswer(userAns: Int) {
    val correctAns = wordBank[curIndex].answer

    val message: String = if (userAns == correctAns) {
        "정답입니다"
    } else {
        "오답입니다"
    }
    Toast.makeText(this, message, Toast.LENGTH_SHORT)
        .show()
}
```

단계 2: TOEIC 단어 퀴즈

- 단계 1: Basic app
  - 화면 구성(ConstraintLayout)
    - 문자열 리소스
  - 문제 은행(data class)
  - 문제 이동(Next, Previous) 기능
  - 정답 확인
- **단계 2: MVC 구조로 변경**
  - **MVC – 문제 은행(ViewModel)**
- 단계 3: 새 화면 추가
  - 화면 구성(LinearLayout + TableLayout)
  - 단어 사전
  - Hint 제공: 2개의 Activity

ViewModel

- ViewModel
  - 특정 Activity에 연결
    - Activity 화면에 보여 줄 데이터를 체계적으로 관리
  - 모델(model) 데이터를 화면에 보여주는(view) 기능
    - 데이터 정의 + 데이터를 access + 데이터 출력
  - 데이터 클래스는 데이터 구조(속성)만 정의하지만
    - ViewModel은 메소드도 함께 정의할 수 있음.
  - Activity 상태 변화에도 **객체의 상태를 보존**
    - **ViewModel의 속성은 상태 변화에도 이전 값을 보존**

- ViewModel은 **androidx.lifecycle** 패키지 일부
  - Activity 와 같은 컴포넌트의 생명주기(lifecycle)를 관찰하고 상태를 고려해 동작
  - Build.gradle에 dependency를 별도도 추가해야 함.

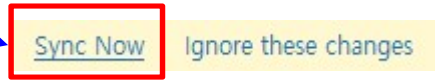
dependencies 속성 추가

lifecycle-extensions API 라이브러리 추가가 필요
➔ **최신 버전은 API(2.3.1) (2021. 6. 기준).**

아티팩트	현재 안정화 버전	다음 버전 후보	베타 버전	알파 버전
lifecycle-*	2.3.1	-	-	2.4.0-alpha02

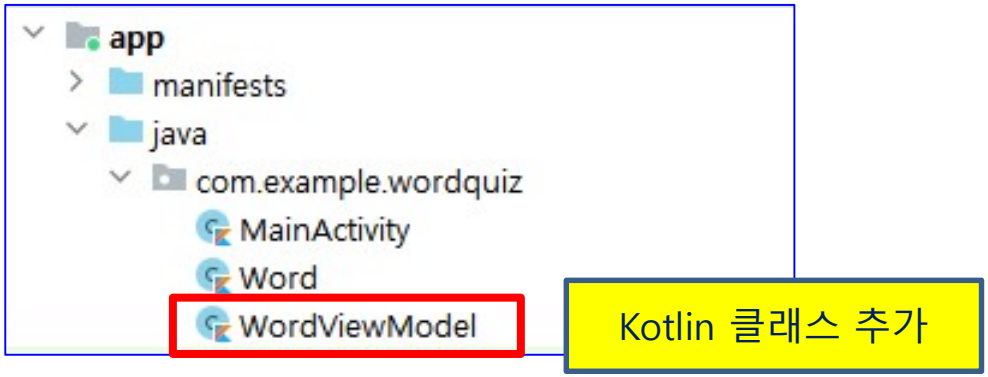
dependencies 블록 안에 아래 문장 추가

```
dependencies {

    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'androidx.core:core-ktx:1.6.0'
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1'
```



Gradle이 동기화하도록 Sync Now 클릭!

단계 2: 프로젝트 구성



- **WordViewModel**
  - 특정 Activity(**MainActivity**)에 연결.
  - 화면에 보여 줄 UI 데이터를 한 곳에서 관리.
    - 모델(**Word**)에서 정의한 구조에 맞춰 데이터 생성.
    - 데이터를 저장하거나 처리하는 기능도 포함.
  - 장치 회전(rotate)과 같은 상태 변화에도 UI 데이터를 보존

WordViewModel

```
class WordViewModel : ViewModel() {
    var curIndex = 0
    private val wordBank = listOf<Word>(...)

    val curAnswer: Int
        get() = wordBank[curIndex].answer
    val curQuestion: String
        get() = wordBank[curIndex].question
    val curNumber1: String
        get() = wordBank[curIndex].number_1
    val curNumber2: String
        get() = wordBank[curIndex].number_2
    val curNumber3: String
        get() = wordBank[curIndex].number_3
    val curNumber4: String
        get() = wordBank[curIndex].number_4

    fun moveToNext() {
        curIndex = (curIndex + 1) % wordBank.size
    }
    fun moveToPrevious() {
        if (curIndex == 0) {
            curIndex = wordBank.size - 1
        } else {
            curIndex -= 1
        }
    }
}
```

curIndex의 접근 제어는 **public**
wordBank는 ViewModel 클래스에서만 access할 수 있도록 **private**으로 선언

6개 속성에 대한 get 메소드 정의
get( ) ➔ 속성 값을 전달

6개 속성의 접근 제어는 모두 public
접근 제어를 명시하지 않으면 public

2개의 메소드 정의
문제 이동(Previous, Next)

Quiz #3 - Activity 코드 수정

초기화는 ViewModel이 처음 사용될 때까지 연기

클래스의 속성 추가

```
class MainActivity : AppCompatActivity() {

    private val wordViewModel: WordViewModel by lazy {
        ViewModelProvider(this).get(WordViewModel::class.java)
    }
```

ViewModel 클래스

```
nextButton.setOnClickListener {  it: View!
    curIndex = (curIndex + 1) % wordBank.size
    updateWordQuiz()
}
prevButton.setOnClickListener {  it: View!
    if (curIndex == 0) {
        curIndex = wordBank.size - 1
    } else {
        curIndex -= 1
    }
    updateWordQuiz()
}
```

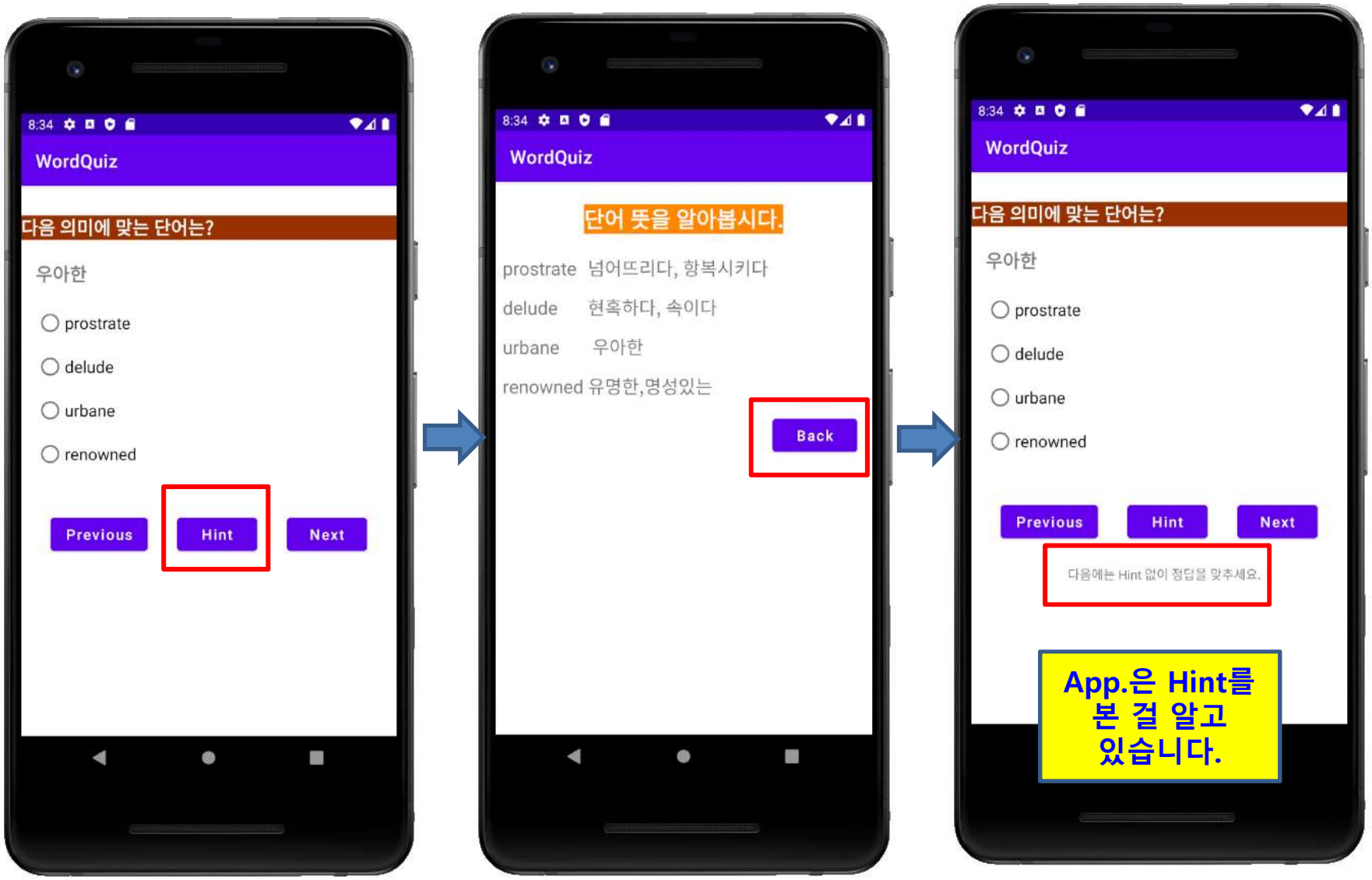
```
nextButton.setOnClickListener {  it: View!
    wordViewModel.moveToNext()
    updateWordQuiz()
}
prevButton.setOnClickListener {  it: View!
    wordViewModel.moveToPrevious()
    updateWordQuiz()
}
```

이 코드를 참고해 MainActivity 의 다른 부분도 수정해 보세요

단계 3: TOEIC 단어 퀴즈

- 단계 1: Basic app
  - 화면 구성(ConstraintLayout)
    - 문자열 리소스
  - 문제 은행(data class)
  - 문제 이동(Next, Previous) 기능
  - 정답 확인
- 단계 2: MVC 구조로 변경
  - MVC – 문제 은행(ViewModel)
- **단계 3: 새 화면 추가**
  - **화면 구성(LinearLayout + TableLayout)**
  - **단어 사전**
  - **Hint 제공: 2개의 Activity**

단계 3: Hint 기능 추가

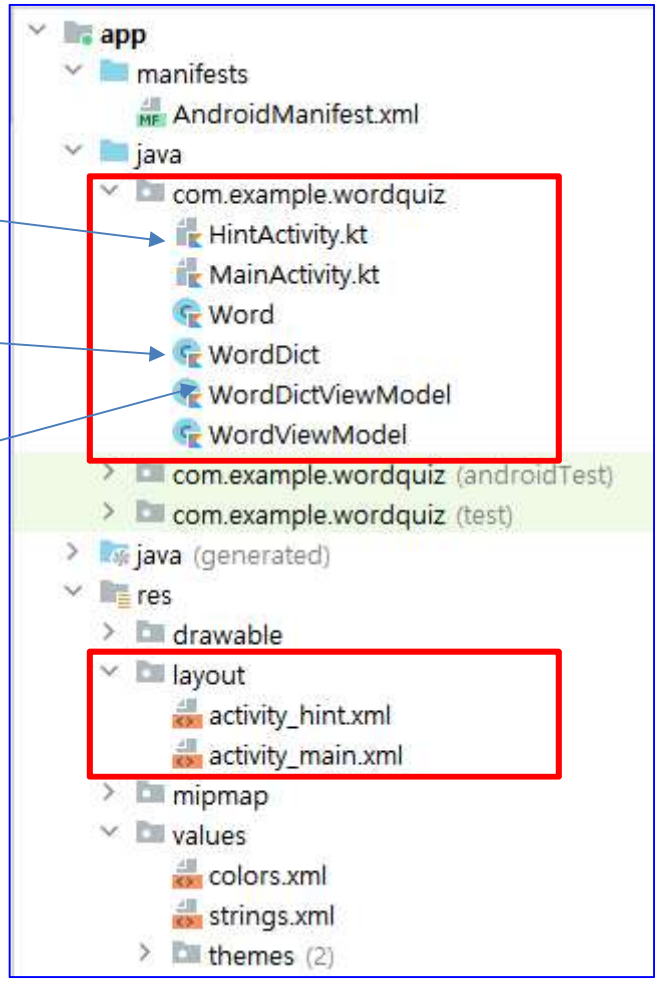


단계 3: Project 구성

Hint 화면 구성
및 제어 class

Hint 제공을 위한
Data class

Hint 처리를 위한
ViewModel class

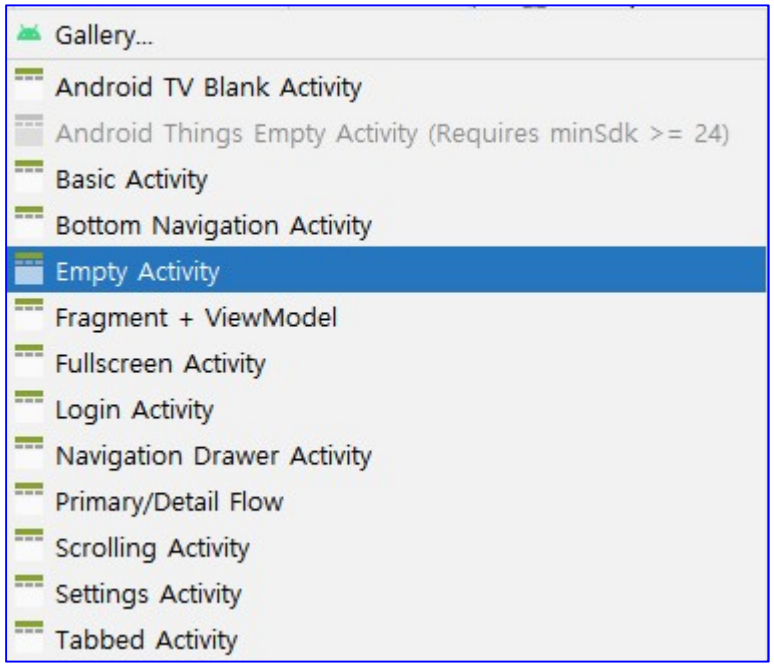


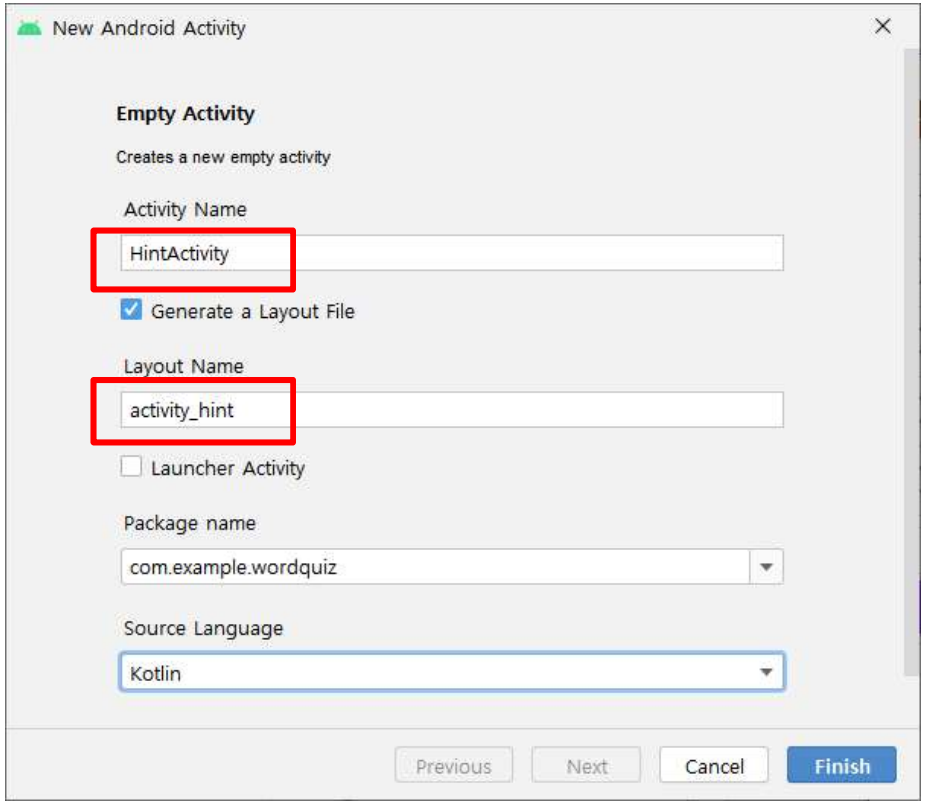
Activity 2개
ViewModel 2개
data 클래스 2개

레이아웃 파일 2개

HintActivity를 레이아웃과 함께 추가

패키지 이름 > 오른쪽 버튼 >
New > **Activity**





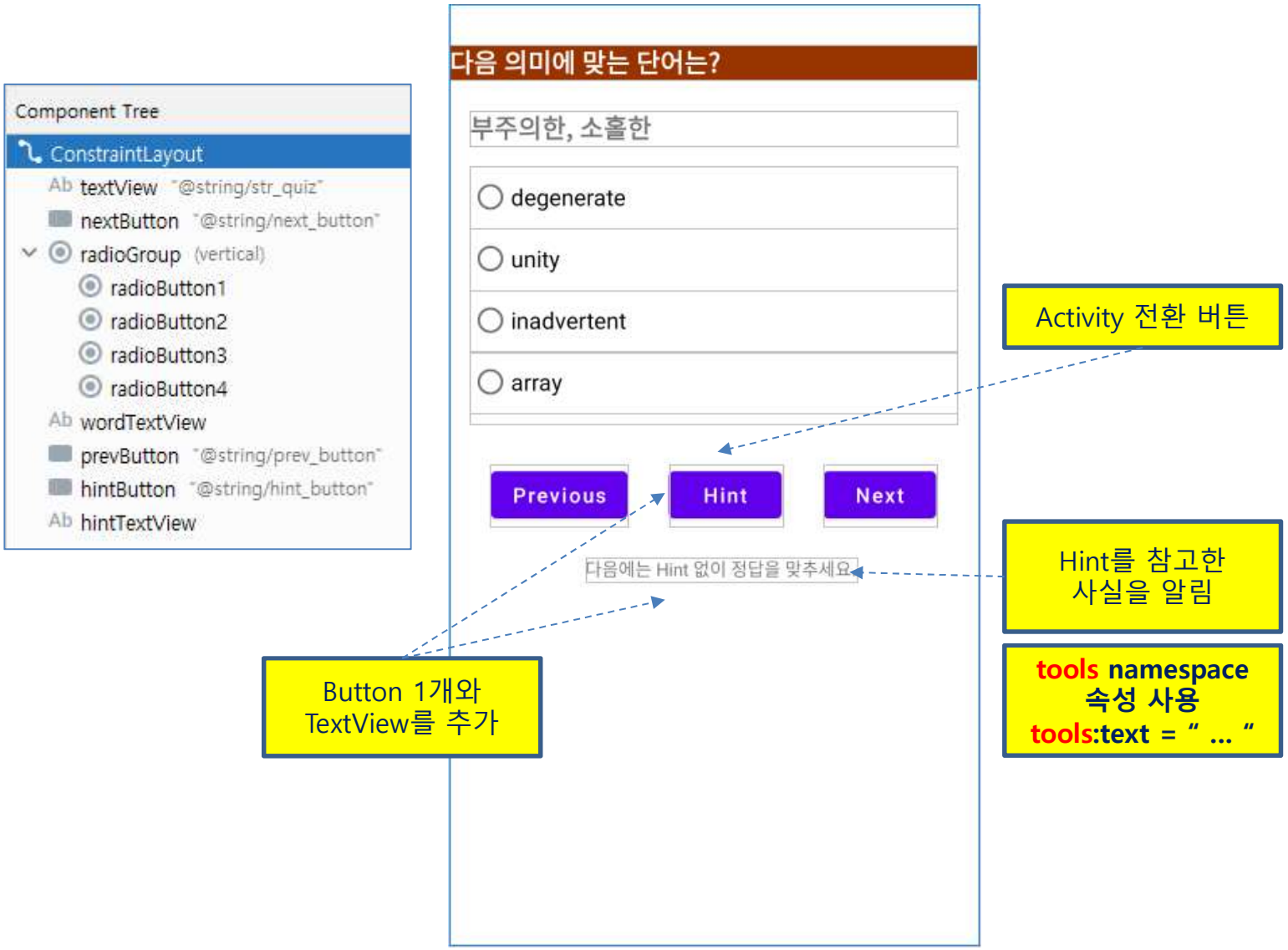
Manifest - HintActivity 추가

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.wordquiz">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="WordQuiz"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.WordQuiz">
        <activity android:name=".HintActivity"></activity>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

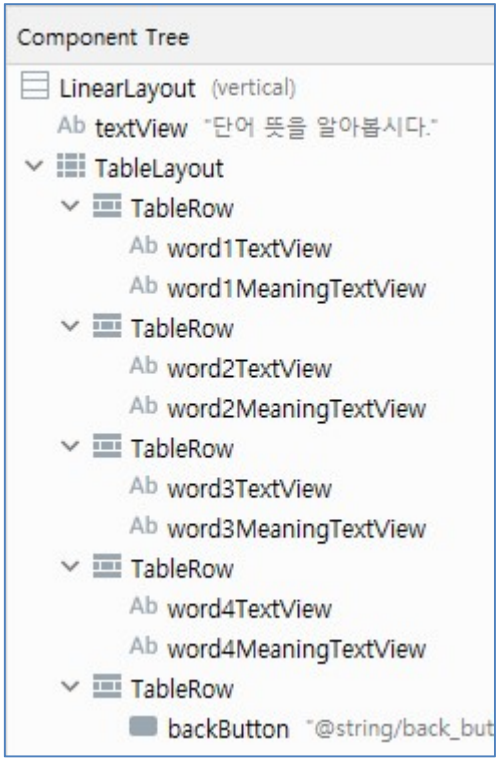
Activity 클래스가 만들어 지면서 **자동 추가됨**

MainActivity가 **Launcher**
➔ App.이 실행될 때 가장 먼저 화면에 나타남.

Layout - activity_main



Layout - activity_hint

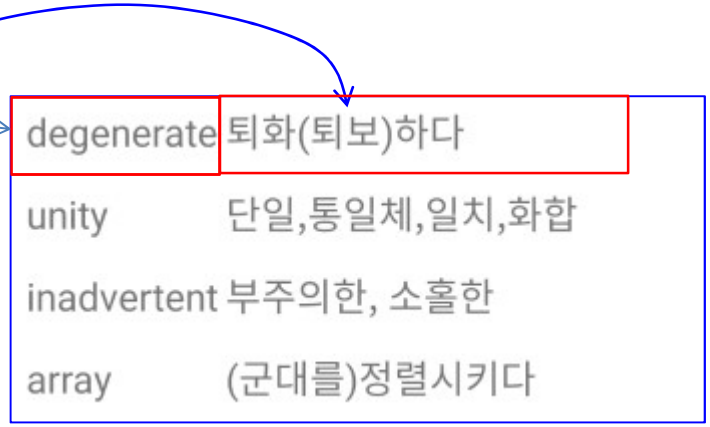


단어와 단어 뜻을 설명

MainActivity로
돌아 감

data class – WordDict

```
package com.example.wordquiz

data class WordDict (
    val word1: String,
    val word1meaning: String,
    val word2: String,
    val word2meaning: String,
    val word3: String,
    val word3meaning: String,
    val word4: String,
    val word4meaning: String
)
```



Quiz #4 - WordDictViewModel

```
class WordDictViewModel : ViewModel()  {

    var curIndexDict = 0

    private val wordDictBank = listOf<WordDict>(
        WordDict("degenerate", "퇴화(퇴보)하다", "unity", "단일,통일체,일치,화합",
            "inadvertent", " 부주의한, 소홀한", "array", "(군대를)정렬시키다"),
        WordDict("vanity", "덧없음, 무상함, 공허", "underhand", "비밀의 음흉한",
            "enslave", " 노예로 만들다", "debilitate", "쇠약하게 하다"),
        WordDict("artisan", "기능공", "sadistic", "남을 학대하는", "glossy",
            " 윤이 나는", "obviate", "(위험·곤란)제거하다"),
        WordDict("prostrate", "넘어뜨리다, 항복시키다", "delude", "현혹하다, 속이다",
            "urbane", " 우아한", "renowned", "유명한,명성있는"),
        WordDict("bereave", "(생명, 희망을)빼앗다", "enliven", "활기있게 하다",
            "occult", " 신비로운 불가사의한", "besiege", "포위(공격)하다")
    )

    val curWord1: String
        get() = wordDictBank[curIndexDict].word1
    val curWord1Meaning: String
        get() = wordDictBank[curIndexDict].word1meaning
```

이 코드를 참고해 WordDictViewModel 의 다른 부분도 수정해 보세요

Quiz #5 - HintActivity

```
class HintActivity : AppCompatActivity() {

    private val wordDictViewModel: WordDictViewModel by lazy {
        ViewModelProvider(this).get(WordDictViewModel::class.java)
    }

    private lateinit var word1TextView: TextView
    private lateinit var word1MeaningTextView: TextView
```

HintActivity에 WordDictViewModel를 연결

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

    word1TextView = findViewById(R.id.word1TextView)
    word1MeaningTextView = findViewById(R.id.word1MeaningTextView)

    wordDictViewModel.curIndexDict = intent.getIntExtra(EXTRA_CUR_INDEX, 0)
    updateWordDict()
}
```

WordDictViewModel의 curIndexDIct 속성에 값 할당

```
private fun updateWordDict() {
    word1TextView.text = wordDictViewModel.curWord1
    word1MeaningTextView.text = wordDictViewModel.curWord1Meaning
}
```

WordDictViewModel의 다른 속성에 값을 가져옴(get)

이 코드를 참고해 HintActivity의 다른 부분도 수정해 보세요

Intent를 사용한 화면 전환 (1/2)

Intent를 사용한 화면 전환 (2/2)

MainActivity

```
import ...

const val EXTRA_CUR_INDEX = "com.example.wordquiz.extra_cur_index"

class MainActivity : AppCompatActivity() {
  hintButton.setOnClickListener {  it: View!
      val intent = Intent(this, HintActivity::class.java)
      intent.putExtra(EXTRA_CUR_INDEX, wordViewModel.curIndex)
      startActivity(intent)
  }
```

클래스 바깥에서 선언한 상수는 Project에 있는 모든 파일에서 공유할 수 있음

HintActivity

```
wordDictViewModel.curIndexDict = intent.getIntExtra(EXTRA_CUR_INDEX, 0)
updateWordDict()

private fun updateWordDict() {
    word1TextView.text = wordDictViewModel.curWord1
    word1MeaningTextView.text = wordDictViewModel.curWord1Meaning

}
```

잠깐! companion object로 구현 (1/2)

- MainActivity는 HintActivity한테만 intent 객체에 데이터(extras)를 실어서 보냄
  - HintActivity에게 전달된 내용을 다른 Activity는 알 필요가 없음.
    - **나만 필요해!**

- **캡슐화(encapsulation)**
  - MainActivity가 HintActivity에게 intent를 요청하는 코드를 별도의 메소드로 만든다.
  - **Companion object (동반 객체) : HintActivity. newIntent (...)**
    - 생성자를 사용해 클래스 인스턴스(instance)를 생성할 필요가 없음.
    - 인스턴스 없이 **클래스 이름을 사용해 동반 객체의 멤버 메소드를 호출**.
      - Java의 정적 메소드와 유사
  - **Companion (동반, 동무 - 함께 움직임)**
    - HintActivity는 **newIntent** ( )를 자신의 멤버(method)인 것처럼 사용.

잠깐! companion object로 구현 (2/2)

HintActivity

```
class HintActivity : AppCompatActivity() {

    private val wordDictViewModel: WordDictViewModel by lazy {...}

    companion object {
        fun newIntent(packageContext: Context, curDictIndex: Int): Intent {
            return Intent(packageContext, HintActivity::class.java).apply {   this: Intent
                putExtra(EXTRA_CUR_INDEX, curDictIndex)
            }
        }
```

MainActivity

```
hintButton.setOnClickListener {   it: View!
    val intent = Intent(this, HintActivity::class.java)
    intent.putExtra(EXTRA_CUR_INDEX, wordViewModel.curIndex)
    startActivity(intent)
}
```

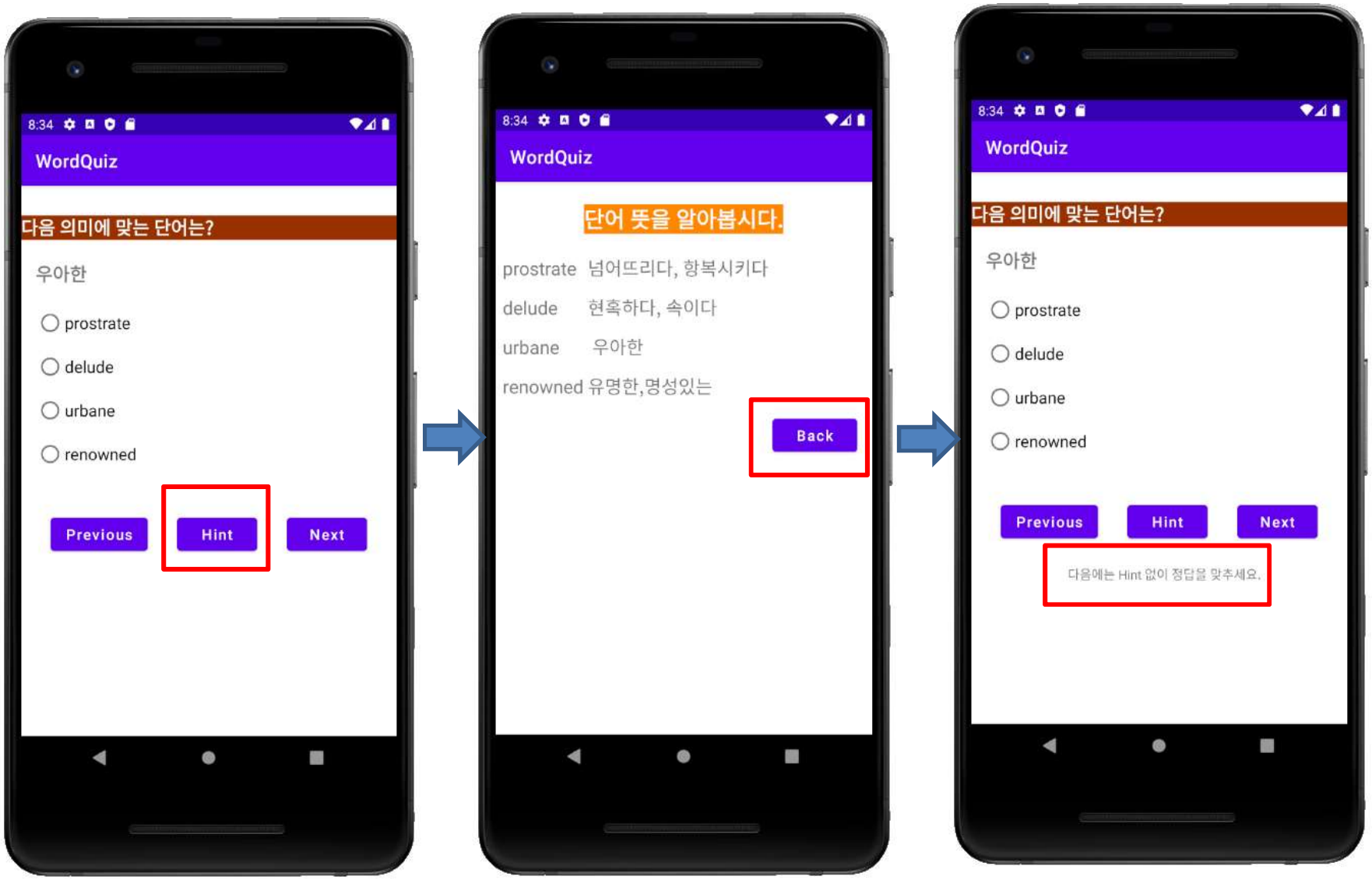
Companion object 의 메소드로 newIntent ( ) 정의
➔ Extras에 실어 보낼 data 정의

MainActivity

```
hintButton.setOnClickListener {   it: View!
    val intent = HintActivity.newIntent(this@MainActivity,
                                        wordViewModel.curIndex)
    startActivity(intent)
}
```

동반 객체의 메소드 호출
HintActivity. newIntent ( )

HintActivity로부터 데이터 수신



HintActivity로부터 데이터 전달받기(1/2)

MainActivity

```
hintButton.setOnClickListener {  it: View!
    val intent = HintActivity.newIntent(this@MainActivity,
                                        wordViewModel.curIndex)
    startForResult.launch(intent)
}
```

수신할 데이터가 있을 경우
호출하는 메소드가 다름

```
private val startForResult = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult())
{ result: ActivityResult ->
    if (result.resultCode == Activity.RESULT_OK) {
        hintReferred = result.data?.getBooleanExtra(EXTRA_HINT_SHOWN, false) ?: false
        if (hintReferred) {
            hintTextView.setText(R.string.hint_message)
        } else {
            hintTextView.text = ""
        }
    }
}
```

수신 데이터를
처리하기 위한 메소드

HintActivity에서 선언한 상수

```
private fun updateWordQuiz() {
    hintReferred = false
    hintTextView.text = ""
}
```

다음 문제를 위해
초기화가 필요

HintActivity로부터 데이터 전달받기(2/2)

HintActivity

```
import ...

const val EXTRA_HINT_SHOWN = "com.example.wordquiz.extra_hint_shown"

class HintActivity : AppCompatActivity() {
```

```
backButton.setOnClickListener { it: View!
    val data = Intent().apply { this: Intent
        putExtra(EXTRA_HINT_SHOWN, true)
    }
    setResult(Activity.RESULT_OK, data)
    finish()
}
```

제대로 처리해서 보냄

Activity 종료(destroy)