

# **LAPORAN PRAKTIKUM**

## **PEMROGRAMAN BERBASIS MOBILE**

### **PERTEMUAN KE-9**



**Disusun Oleh :**

**NAMA** : Raden Isnawan Argi Aryasatya  
**NIM** : 195410257  
**JURUSAN** : Informatika  
**JENJANG** : S1  
**KELAS** : 5

**Laboratorium Terpadu**  
**Sekolah Tinggi Manajemen Informatika Komputer**  
**AKAKOM**  
**YOGYAKARTA**

**2021**

## **PERTEMUAN KE-9**

### **(LIVEDATA DAN LIVEDATA OBSERVER)**

#### **TUJUAN**

Mahasiswa mampu menggunakan LiveData dan LiveData observer dalam aplikasi

#### **DASAR TEORI**

### **1. Selamat datang**

Di codelab sebelumnya, Anda menggunakan ViewModel di aplikasi GuessTheWord untuk memungkinkan data aplikasi bertahan dari perubahan konfigurasi perangkat. Dalam codelab ini, Anda mempelajari cara mengintegrasikan LiveData dengan data di kelas ViewModel. LiveData, yang merupakan salah satu Komponen Arsitektur Android, memungkinkan Anda membuat objek data yang memberi tahu tampilan saat database yang mendasarinya berubah. Untuk menggunakan kelas LiveData, Anda menyiapkan pengamat atau "observer" (misalnya, aktivitas atau fragmen) yang mengamati perubahan dalam data aplikasi. LiveData peka terhadap siklus proses, jadi LiveData hanya mengupdate pengamat komponen aplikasi yang berada dalam status siklus proses aktif.

#### **Apa Yang Harus Anda Ketahui**

- Cara membuat aplikasi Android dasar di Kotlin.
- Cara menavigasi di antara tujuan aplikasi Anda.
- Daur hidup aktivitas dan fragmen.
- Cara menggunakan objek ViewModel di aplikasi Anda.
- Cara membuat objek ViewModel menggunakan antarmuka ViewModelProvider.Factory.

#### **Apa yang akan Anda pelajari**

- Apa yang membuat objek LiveData berguna.
- Bagaimana menambahkan LiveData ke data yang disimpan di ViewModel.
- Kapan dan bagaimana menggunakan MutableLiveData.
- Bagaimana menambahkan metode pengamat untuk mengamati perubahan di LiveData.
- Cara merangkum LiveData menggunakan properti pendukung.
- Cara berkomunikasi antara pengontrol UI dan ViewModel yang sesuai.

#### **Apa yang akan Anda lakukan**

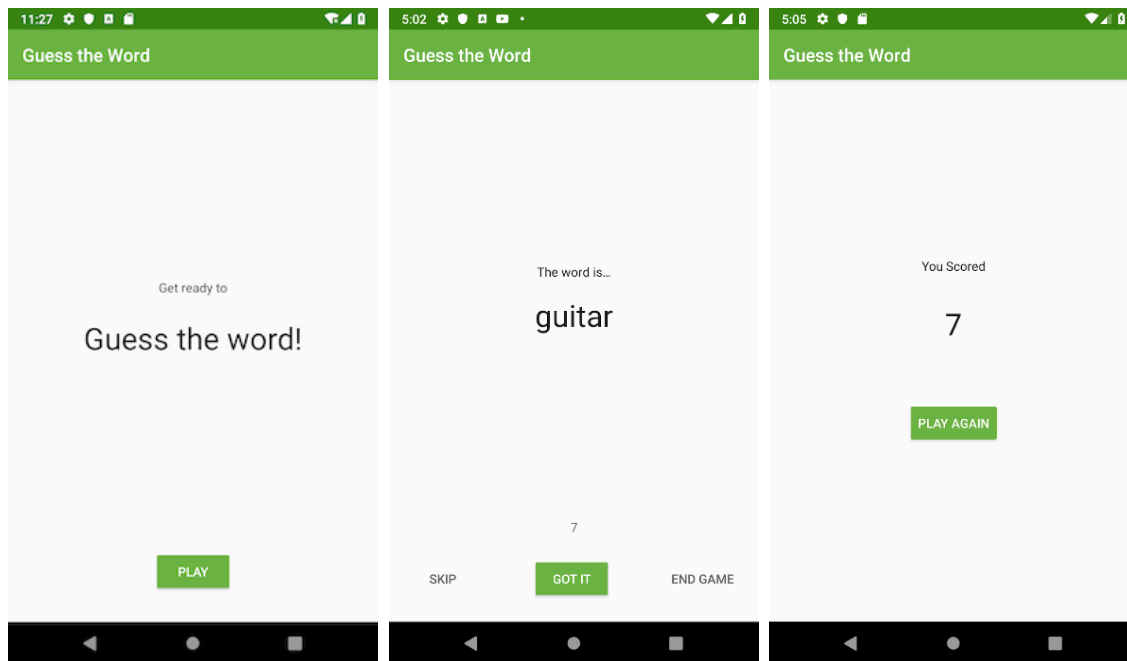
- Gunakan LiveData untuk kata dan skor di aplikasi GuessTheWord.
- Tambahkan pengamat yang memperhatikan ketika kata atau skor berubah.
- Perbarui tampilan teks yang menampilkan nilai yang diubah.
- Gunakan pola pengamat LiveData untuk menambahkan acara yang sudah selesai game.
- Terapkan tombol Play Again.

### **2. Ikhtisar Aplikasi**

Dalam modul sebelumnya, Anda mengembangkan aplikasi GuessTheWord, dimulai dengan kode permulaan. GuessTheWord adalah permainan gaya sandiwara dua pemain, di mana para pemain berkolaborasi untuk mencapai skor setinggi mungkin.

Dalam modul ini, Anda meningkatkan aplikasi GuessTheWord dengan menambahkan peristiwa untuk mengakhiri game saat pengguna menelusuri semua kata di aplikasi. Anda juga menambahkan tombol Play Again di fragmen skor, sehingga pengguna dapat memainkan game lagi.

Seperti gambar dibawah ini: (gambar dibawah ini hanya contoh dari modul, bukan dari emulator saya)



Layar Title

Layar Game

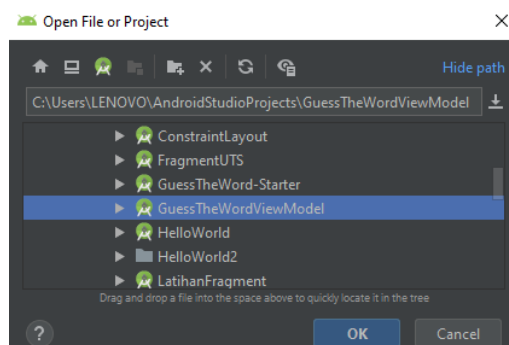
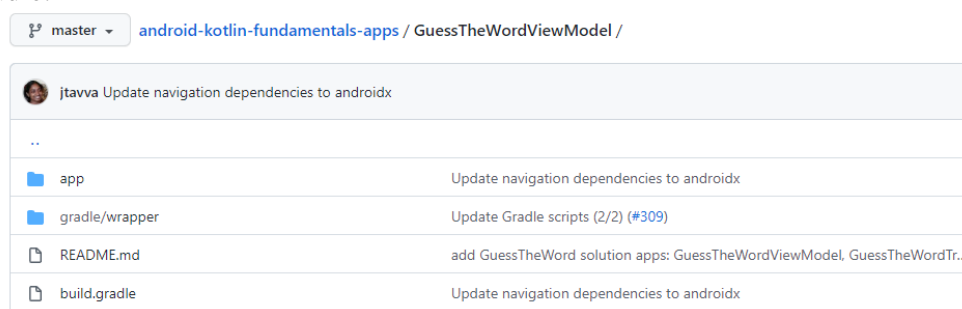
Layar Score

## PRAKTIK

### 3. Tugas: Memulai

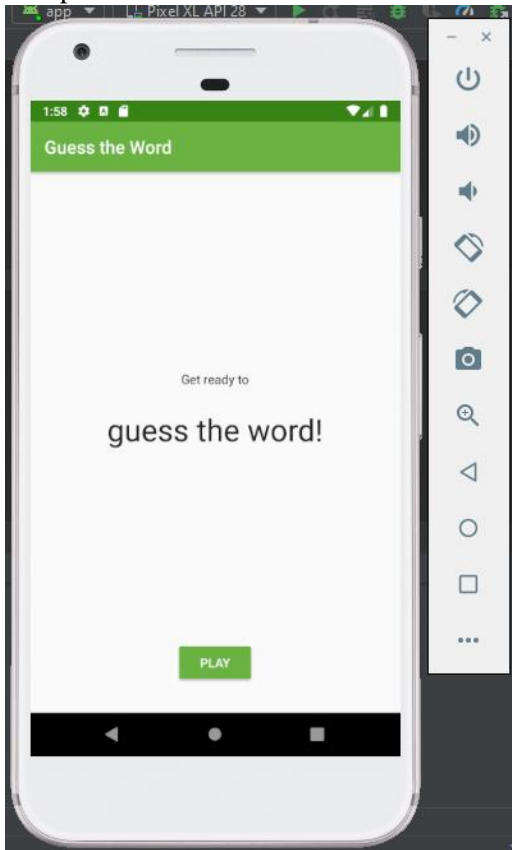
Dalam tugas ini, Anda mencari dan menjalankan kode awal untuk codelab ini. Anda dapat menggunakan aplikasi GuessTheWord yang Anda buat di codelab sebelumnya sebagai kode permulaan, atau Anda dapat mengunduh aplikasi permulaan.

1. (Opsional) Jika Anda tidak menggunakan kode dari codelab sebelumnya, unduh kode awal (GuessTheWordViewModel) untuk codelab ini. Buka zip kodenya, dan buka proyek di Android Studio.

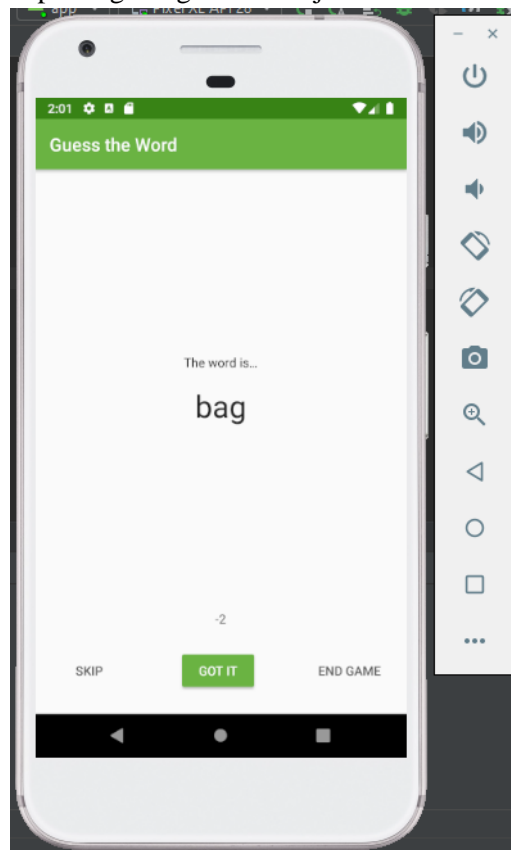


2. Jalankan aplikasi dan mainkan gamenya.

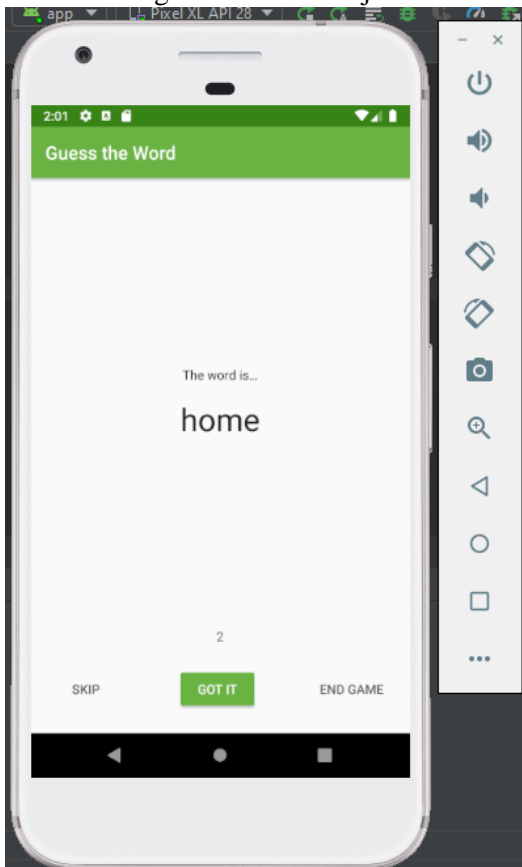
Tampilan awal



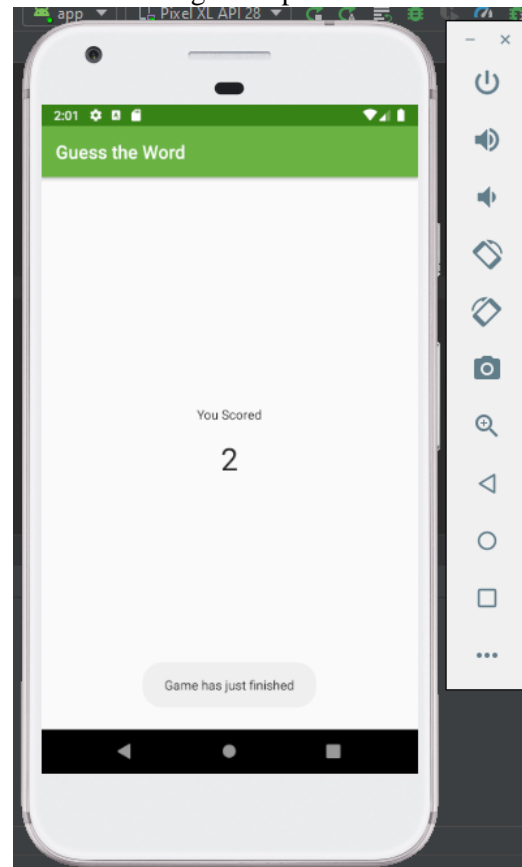
Skip mengurangi skor menjadi -2



Got it meningkatkan skor menjadi 2



End Game mengakhiri permainan



3. Perhatikan bahwa tombol Skip menampilkan kata berikutnya dan mengurangi skor satu per satu, dan tombol Got It menampilkan kata berikutnya dan meningkatkan skor satu per satu. Tombol End Game mengakhiri permainan.

---

## 4. Tugas: Tambahkan LiveData ke GameViewModel

LiveData adalah kelas pemegang data yang dapat diamati yang peka terhadap siklus proses. Misalnya, Anda dapat menggabungkan LiveData dengan skor saat ini di aplikasi GuessTheWord. Di codelab ini, Anda mempelajari tentang beberapa karakteristik LiveData:

- LiveData dapat diamati, yang berarti bahwa pengamat diberi tahu saat data yang dipegang oleh objek LiveData berubah.
- LiveData menyimpan data; LiveData adalah pembungkus yang dapat digunakan dengan data apa pun
- LiveData sadar siklus proses. Saat Anda memasang pengamat ke LiveData, pengamat dikaitkan dengan LifecycleOwner (biasanya Aktivitas atau Fragment). LiveData hanya memperbarui pengamat yang berada dalam status siklus hidup aktif seperti STARTED atau RESUMED.

Dalam tugas ini, Anda mempelajari cara menggabungkan tipe data apa pun ke dalam objek LiveData dengan mengonversi skor saat ini dan data kata saat ini di GameViewModel ke LiveData. Di tugas selanjutnya, Anda menambahkan pengamat ke objek LiveData ini dan mempelajari cara mengamati LiveData.

### Langkah 1: Ubah score dan word untuk menggunakan LiveData

1. Di bawah paket screens/game, buka file GameViewModel.
2. Ubah tipe variabel score dan word ke MutableLiveData.

MutableLiveData adalah LiveData yang nilainya dapat diubah. MutableLiveData adalah kelas umum, jadi Anda perlu menentukan jenis data yang dimilikinya.

```
// The current word
val word = MutableLiveData<String>()

// The current score
val score = MutableLiveData<Int>()
```

3. Di GameViewModel, di dalam blok init, inisialisasi score dan word. Untuk mengubah nilai variabel LiveData, Anda menggunakan metode setValue () pada variabel. Di Kotlin, Anda bisa memanggil setValue () menggunakan properti value.

```
init {
    word.value = ""
    score.value = 0
    Log.i( tag: "GameViewModel", msg: "GameViewModel created!")
    resetList()
    nextWord()
}
```

### Langkah 2: Perbarui referensi objek LiveData

Variabel score dan word sekarang berjenis LiveData. Pada langkah ini, Anda mengubah referensi ke variabel ini, menggunakan properti value.

1. Di GameViewModel, di metode onSkip (), ubah score menjadi score.value. Perhatikan kesalahan tentang skor yang mungkin menjadi nol. Anda memperbaiki kesalahan ini selanjutnya.
2. Untuk mengatasi kesalahan, tambahkan pemeriksaan null ke score.value di onSkip (). Kemudian panggil fungsi minus () pada score, yang melakukan pengurangan dengan keamanan nol.

```
fun onSkip() {
    score.value = (score.value)?.minus( other: 1)
    nextWord()
}
```

3. Perbarui metode inCorrect () dengan cara yang sama: tambahkan pemeriksaan null ke variabel score dan gunakan fungsi plus ().

```
fun onCorrect() {
    score.value = (score.value)?.plus( other: 1)
    nextWord()
}
```

4. Di GameViewModel, di dalam metode nextWord (), ubah referensi word menjadi word.value.

```
private fun nextWord() {
    //Select and remove a word from the list
    if (!wordList.isEmpty()) {
        //Select and remove a word from the list
        word.value = wordList.removeAt( index: 0)
    }
}
```

5. Di GameFragment, di dalam metode updateWordText (), ubah referensi ke viewModel.word menjadi viewModel.word.value.

```
private fun updateWordText() {
    binding.wordText.text = viewModel.word.value
}
```

6. Dalam GameFragment, di dalam metode updateScoreText (), ubah referensi ke viewModel.score menjadi viewModel.score.value.

```
private fun updateScoreText() {
    binding.scoreText.text = viewModel.score.value.toString()
}
```

7. Di GameFragment, di dalam metode gameFinished (), ubah referensi ke viewModel.score menjadi viewModel.score.value. Tambahkan pemeriksaan keamanan null yang diperlukan

```
private fun gameFinished() {
    Toast.makeText(activity, text: "Game has just finished", Toast.LENGTH_SHORT).show()
    val action = GameFragmentDirections.actionGameToScore()
    action.score = viewModel.score.value?:0
    NavHostFragment.findNavController( fragment: this).navigate(action)
}
```

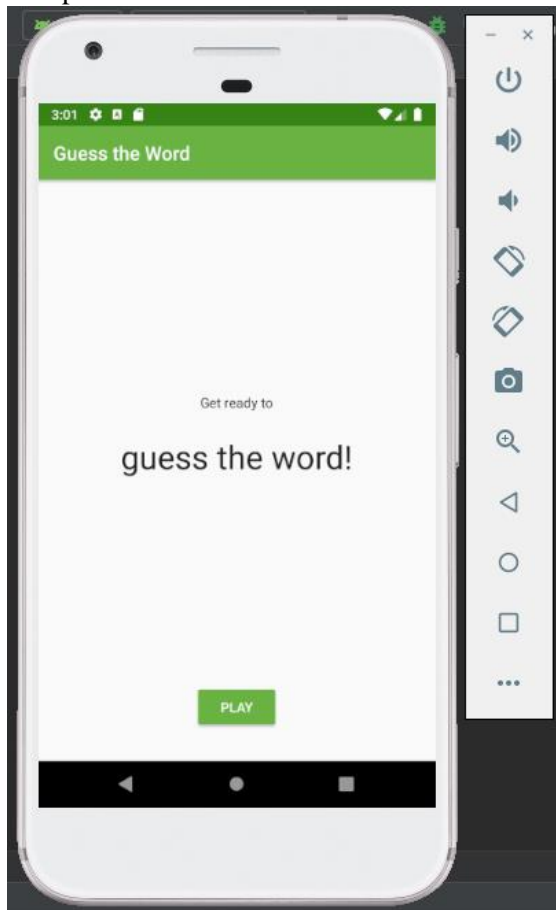
8. Pastikan tidak ada kesalahan dalam kode Anda. Kompilasi dan jalankan aplikasi Anda. Fungsionalitas aplikasi harus sama seperti sebelumnya.

(screenshot aplikasi di halaman selanjutnya)

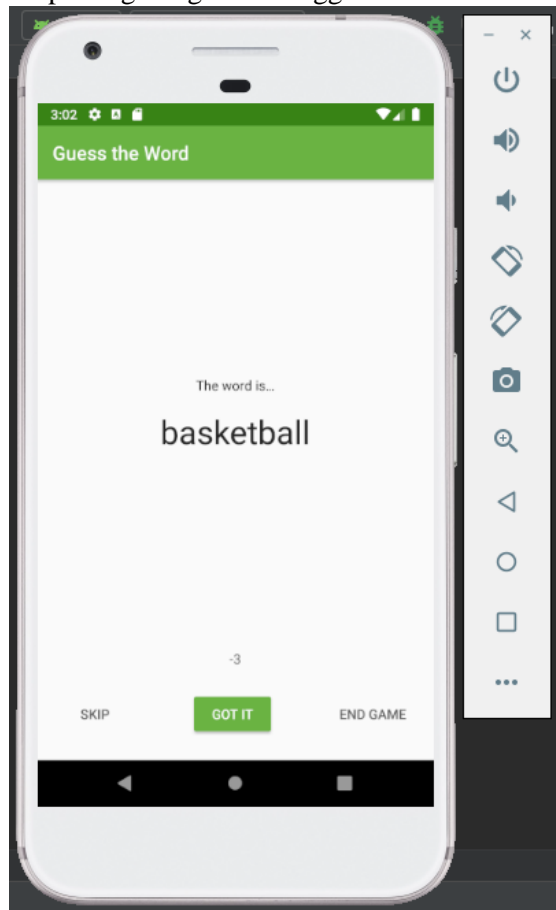
(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)

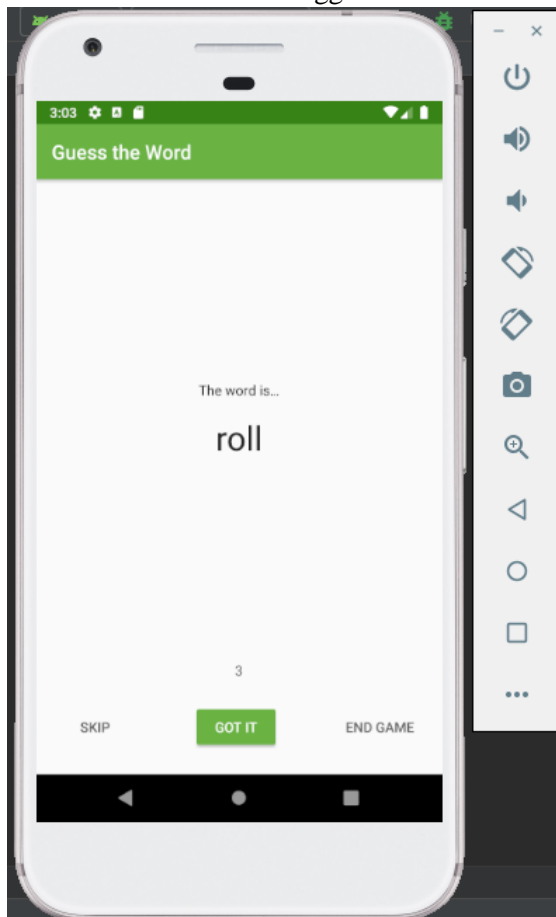
Tampilan awal



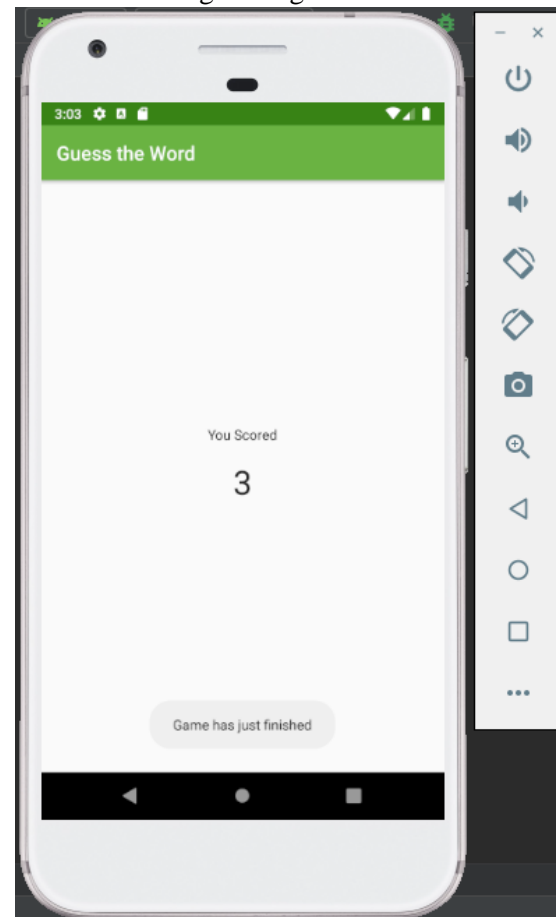
Skip mengurangi skor hingga -3



Get It menambah skor hingga 3



End Game mengakhiri game



## 5. Tugas: Lampirkan pengamat ke objek LiveData

Tugas ini terkait erat dengan tugas sebelumnya, di mana Anda mengonversi data score dan word menjadi objek LiveData. Dalam tugas ini, Anda melampirkan objek Observer ke objek LiveData tersebut. Anda akan menggunakan tampilan fragmen (viewLifecycleOwner) sebagai LifecycleOwner.

1. Dalam GameFragment, di dalam metode onCreateView (), lampirkan objek Observer ke objek LiveData untuk score saat ini, viewModel.score. Gunakan metode observe (), dan letakkan kode setelah inisialisasi viewModel. Gunakan ekspresi lambda untuk menyederhanakan kode. (Ekspresi lambda adalah fungsi anonim yang tidak dideklarasikan, tetapi segera diteruskan sebagai ekspresi.)

```
viewModel.score.observe(viewLifecycleOwner, Observer { newScore ->
})
```

Selesaikan referensi ke Observer. Untuk melakukan ini, klik Observer, tekan Alt + Enter (Option + Enter di Mac), dan impor androidx.lifecycle.Observer.

```
viewModel.score.observe(viewLifecycleOwner, Observer { newScore ->
})

// Imports
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModel
```

2. Observer yang baru saja Anda buat menerima event (kejadian) saat data yang dipegang oleh objek LiveData yang diamati berubah. Di dalam pengamat, perbarui TextView skor dengan skor baru.

```
/** Setting up LiveData observation relationship */
viewModel.score.observe(viewLifecycleOwner, Observer { newScore ->
    binding.scoreText.text = newScore.toString()
})
```

3. Lampirkan objek Observer ke objek word LiveData saat ini. Lakukan dengan cara yang sama seperti Anda memasang objek Observer ke score saat ini.

```
/** Setting up LiveData observation relationship */
viewModel.word.observe(viewLifecycleOwner, Observer { newWord ->
    binding.wordText.text = newWord
})
```

Ketika nilai score atau word berubah, score atau word yang ditampilkan di layar sekarang diperbarui secara otomatis.

4. Di GameFragment, hapus metode updateWordText () dan updateScoreText (), dan semua referensinya. Anda tidak membutuhkannya lagi, karena tampilan teks diperbarui dengan metode pengamat LiveData.

```
binding.correctButton.setOnClickListener { onCorrect() }
binding.skipButton.setOnClickListener { onSkip() }
binding.endGameButton.setOnClickListener { onEndGame() }
updateScoreText()
updateWordText()
return binding.root

private fun updateWordText() {
    binding.wordText.text = viewModel.word.value
}

private fun updateScoreText() {
    binding.scoreText.text = viewModel.score.value.toString()
}
```

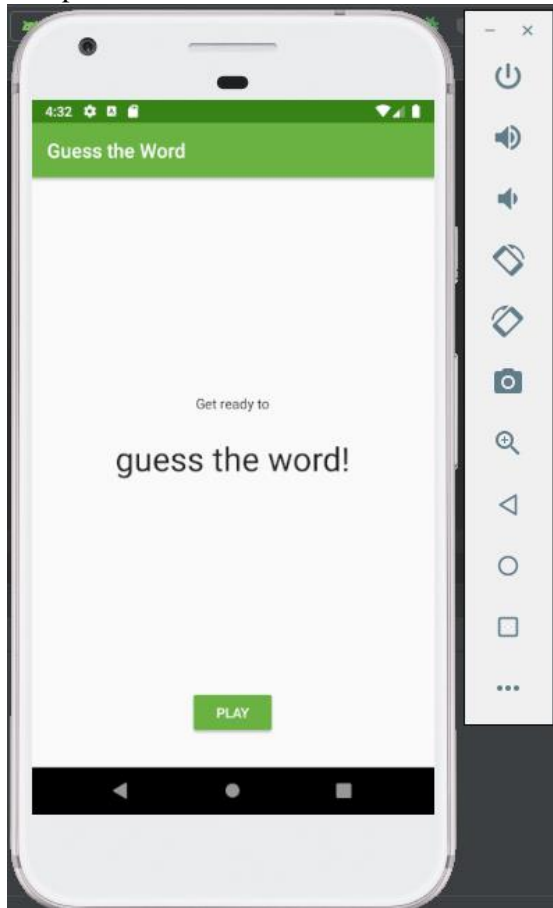
```
private fun onSkip() {
    viewModel.onSkip()
    updateWordText()
    updateScoreText()
}

private fun onCorrect() {
    viewModel.onCorrect()
    updateScoreText()
    updateWordText()
}
```

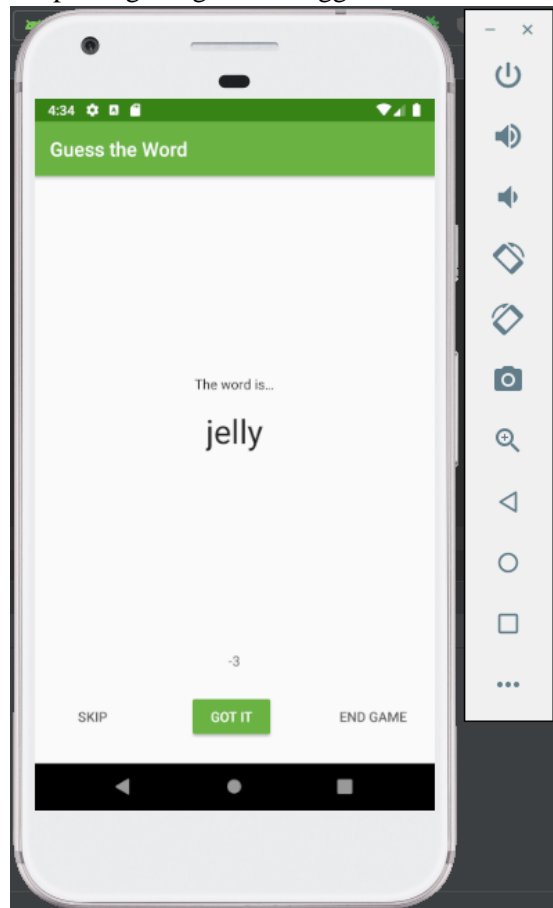
5. Jalankan aplikasi Anda. Aplikasi game Anda seharusnya berfungsi persis seperti sebelumnya, tetapi sekarang menggunakan pengamat LiveData dan LiveData.



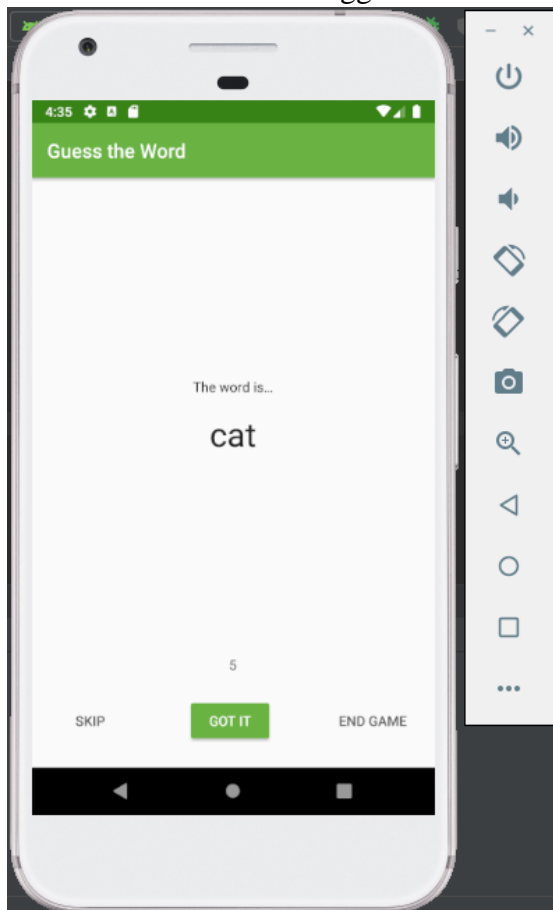
Tampilan awal



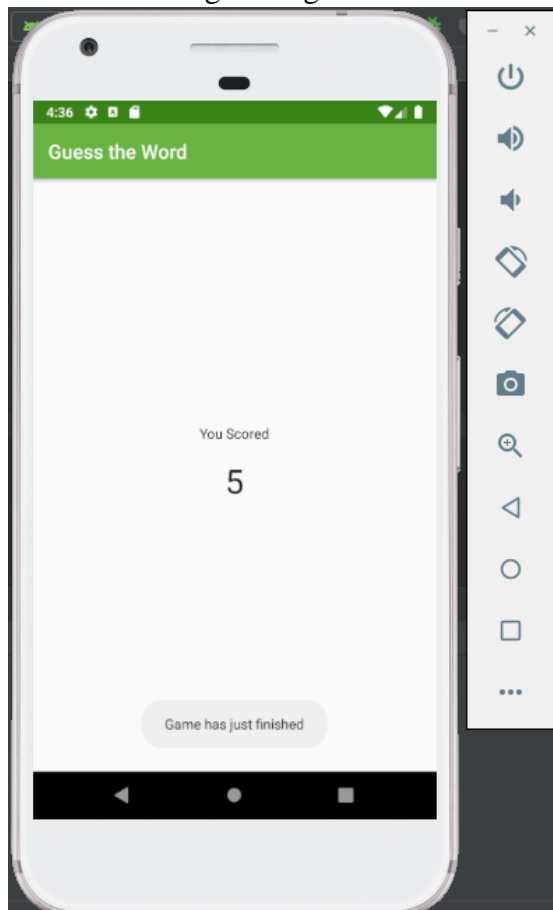
Skip mengurangi skor hingga -3



Get it menambah skor hingga 5



End Game mengakhiri game



---

## 6. Tugas: Mengemas LiveData

Enkapsulasi adalah cara untuk membatasi akses langsung ke beberapa bidang objek. Saat Anda mengemas objek, Anda mengekspos sekumpulan metode publik yang mengubah field internal private. Menggunakan enkapsulasi, Anda mengontrol bagaimana kelas lain memanipulasi field internal ini.

Dalam kode Anda saat ini, setiap kelas eksternal dapat mengubah skor dan variabel kata menggunakan properti nilai, misalnya menggunakan `viewModel.score.value`. Mungkin tidak masalah dalam aplikasi yang Anda kembangkan dalam codelab ini, tetapi dalam aplikasi produksi, Anda ingin mengontrol data di objek `ViewModel`.

Hanya `ViewModel` yang harus mengedit data di aplikasi Anda. Tapi pengontrol UI perlu membaca data, jadi bidang data tidak bisa sepenuhnya pribadi. Untuk merangkum data aplikasi Anda, Anda menggunakan objek `MutableLiveData` dan `LiveData`.

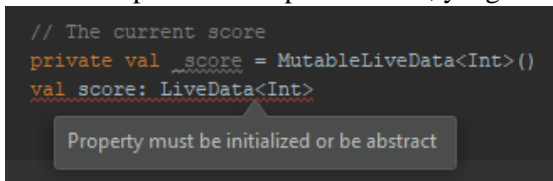
### MutableLiveData vs LiveData:

- Data dalam objek `MutableLiveData` bisa diubah, seperti namanya. Di dalam `ViewModel`, data harus dapat diedit, sehingga menggunakan `MutableLiveData`.
- Data dalam objek `LiveData` dapat dibaca, tetapi tidak dapat diubah. Dari luar `ViewModel`, data harus dapat dibaca, tetapi tidak dapat diedit, sehingga data harus diekspos sebagai `LiveData`.

Untuk menjalankan strategi ini, Anda menggunakan properti dukungan Kotlin. Properti pendukung memungkinkan Anda mengembalikan sesuatu dari pengambil selain objek yang tepat. Dalam tugas ini, Anda mengimplementasikan properti backing untuk `score` dan objek `word` di aplikasi `GuessTheWord`.

### Tambahkan properti pendukung untuk `score` dan `word`:

1. Di `GameViewModel`, buat objek `score` saat ini menjadi private.
2. Untuk mengikuti konvensi penamaan yang digunakan di properti pendukung, ubah `score` menjadi `_score`. Properti `_score` sekarang adalah versi skor game yang bisa diubah, untuk digunakan secara internal.
3. Buat versi publik dari tipe `LiveData`, yang disebut `score`.

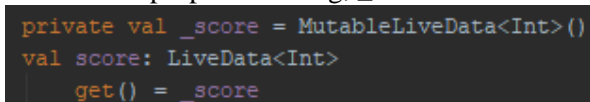


```
// The current score
private val _score = MutableLiveData<Int>()
val score: LiveData<Int>
```

Property must be initialized or be abstract

4. Anda melihat kesalahan inisialisasi. Kesalahan ini terjadi karena di dalam `GameFragment`, `score` adalah referensi `LiveData`, dan `score` tidak dapat lagi mengakses setter-nya. Untuk mempelajari lebih lanjut tentang pengambil dan penyetel di Kotlin, lihat [Getter dan Setter](#).

Untuk mengatasi kesalahan, override metode `get ()` untuk objek `score` di `GameViewModel` dan kembalikan properti backing, `_score`.



```
private val _score = MutableLiveData<Int>()
val score: LiveData<Int>
    get() = _score
```

5. Di `GameViewModel`, ubah referensi `score` ke versi internalnya yang dapat berubah, `_score`.

```
init {
    word.value = ""
    _score.value = 0
    Log.i( tag: "GameViewModel", msg: "GameViewModel created!")
    resetList()
    nextWord()
}
```

```
/** Methods for updating the UI */
fun onSkip() {
    _score.value = (score.value)?.minus( other: 1)
    nextWord()
}
fun onCorrect() {
    _score.value = (score.value)?.plus( other: 1)
    nextWord()
}
```

- Ubah nama objek word menjadi \_word dan tambahkan properti pendukung untuknya, seperti yang Anda lakukan untuk objek score.

```
// The current word
private val _word = MutableLiveData<String>()
val word: LiveData<String>
    get() = _word
```

```
init {
    _word.value = ""
    _score.value = 0
}
```

```
private fun nextWord() {
    //Select and remove a word from the list
    if (!wordList.isEmpty()) {
        //Select and remove a word from the list
        _word.value = wordList.removeAt( index: 0)
    }
}
```

Kerja bagus, Anda telah mengemas word dan score objek LiveData.

## 7. Tugas: Menambahkan event game selesai

Aplikasi Anda saat ini menavigasi ke layar skor ketika pengguna mengetuk tombol End Game. Anda juga ingin aplikasi menavigasi ke layar skor ketika para pemain telah menelusuri melalui semua kata. Setelah pemain selesai dengan kata terakhir, Anda ingin permainan berakhir secara otomatis sehingga pengguna tidak perlu mengetuk tombol.

Untuk mengimplementasikan fungsionalitas ini, Anda memerlukan kejadian (event) yang akan dipicu dan dikomunikasikan ke fragmen dari ViewModel ketika semua kata telah ditampilkan. Untuk melakukan ini, Anda menggunakan pola pengamat LiveData untuk membuat model acara yang diselesaikan game.

### Pola observer

Pola observer (pengamat) adalah pola desain perangkat lunak. Ini menentukan komunikasi antara objek: yang dapat diamati ("subjek" pengamatan) dan pengamat. Observable adalah objek yang memberi tahu pengamat

tentang perubahan statusnya. Dalam kasus LiveData di aplikasi ini, yang dapat diamati (subjek) adalah objek LiveData, dan pengamat adalah metode dalam pengontrol UI, seperti fragmen. Perubahan status terjadi setiap kali data yang dibungkus di dalam LiveData berubah. Kelas LiveData sangat penting dalam berkomunikasi dari ViewModel ke fragmen.

### Langkah 1: Gunakan LiveData untuk mendeteksi acara yang selesai game

Dalam tugas ini, Anda menggunakan pola pengamat LiveData untuk membuat model acara yang diselesaikan game.

1. Di GameViewModel, buat objek Boolean MutableLiveData bernama `_eventGameFinish`. Objek ini akan mengadakan acara yang sudah selesai game.
2. Setelah menginisialisasi objek `_eventGameFinish`, buat dan inisialisasi properti backing yang disebut `eventGameFinish`.

```
// Event which triggers the end of the game
private val _eventGameFinish = MutableLiveData<Boolean>()
val eventGameFinish: LiveData<Boolean>
    get() = _eventGameFinish
```

3. Di GameViewModel, tambahkan metode `onGameFinish()`. Dalam metode ini, setel event untuk game selesai, `eventGameFinish`, ke `true`.

```
/** Method for the game completed event */
fun onGameFinish() {
    _eventGameFinish.value = true
}
```

4. Dalam GameViewModel, di dalam metode `nextWord()`, akhiri permainan dari daftar kata kosong.

```
private fun nextWord() {
    if (wordList.isEmpty()) {
        onGameFinish()
    } else {
        //Select and remove a _word from the
        _word.value = wordList.removeAt(index)
    }
}
```

5. Di GameFragment, di dalam `onCreateView()`, setelah menginisialisasi `viewModel`, lampirkan pengamat ke `eventGameFinish`. Gunakan metode `observe()`. Di dalam fungsi lambda, panggil metode `gameFinished()`.

```
// Observer for the Game finished event
viewModel.eventGameFinish.observe(viewLifecycleOwner, Observer<Boolean> { hasFinished ->
    if (hasFinished) gameFinished()
}))
```

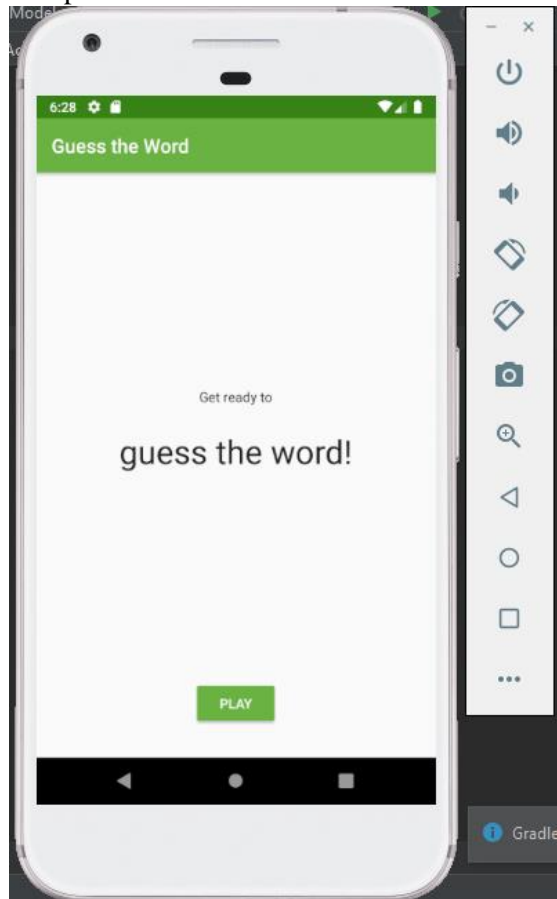
6. Jalankan aplikasi Anda, mainkan game, dan telusuri semua kata. Aplikasi menavigasi ke layar skor secara otomatis, sebagai ganti tetap berada di fragmen game sampai Anda mengetuk Akhiri Game. Setelah daftar kata kosong, `eventGameFinish` disetel, metode pengamat terkait dalam fragmen game dipanggil, dan aplikasi menavigasi ke fragmen layar.

Ada 21 kata sehingga kita telusuri hingga skor 21 sampai aplikasi menavigasi ke layar skor

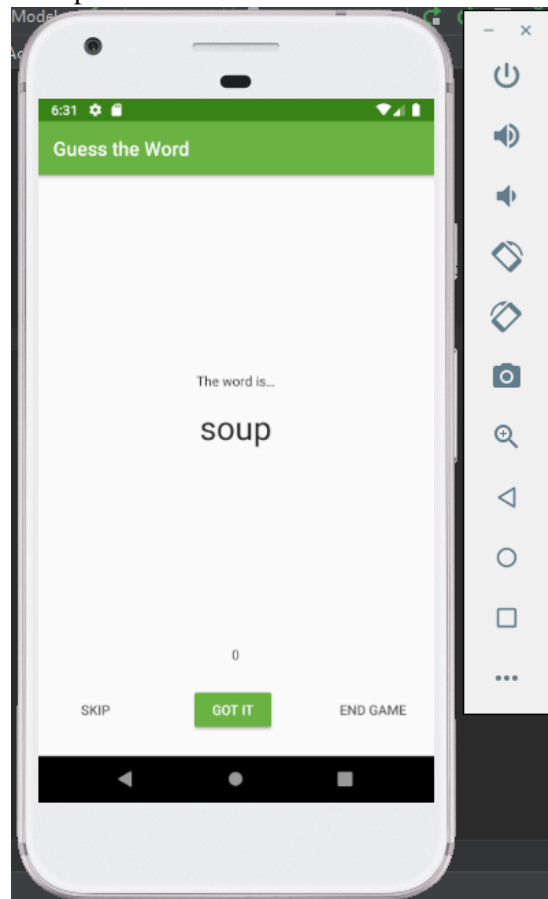
(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)

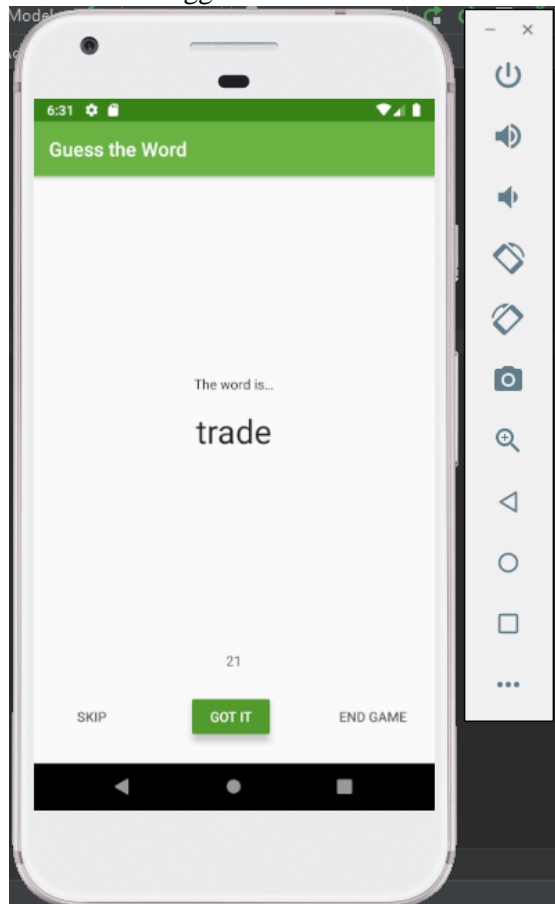
Tampilan awal



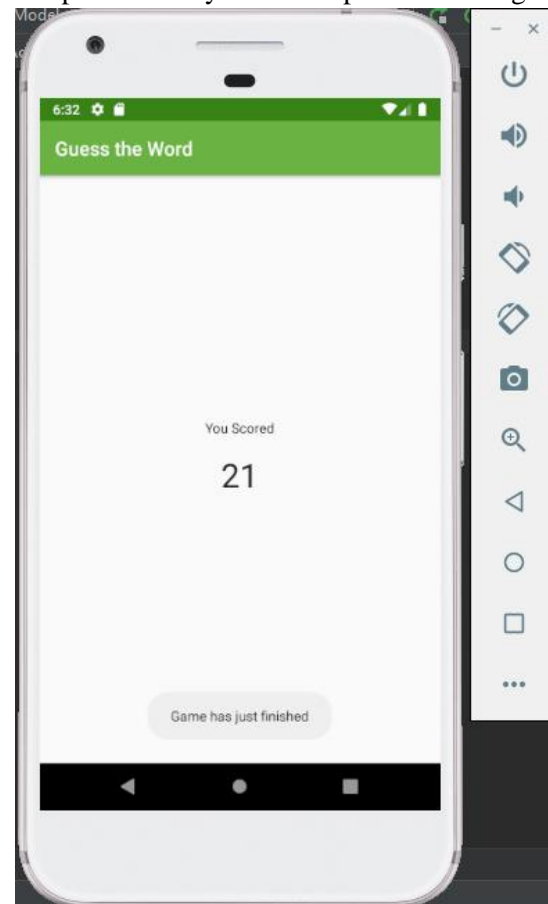
Kata pertama



Klik Got It hingga kata terakhir



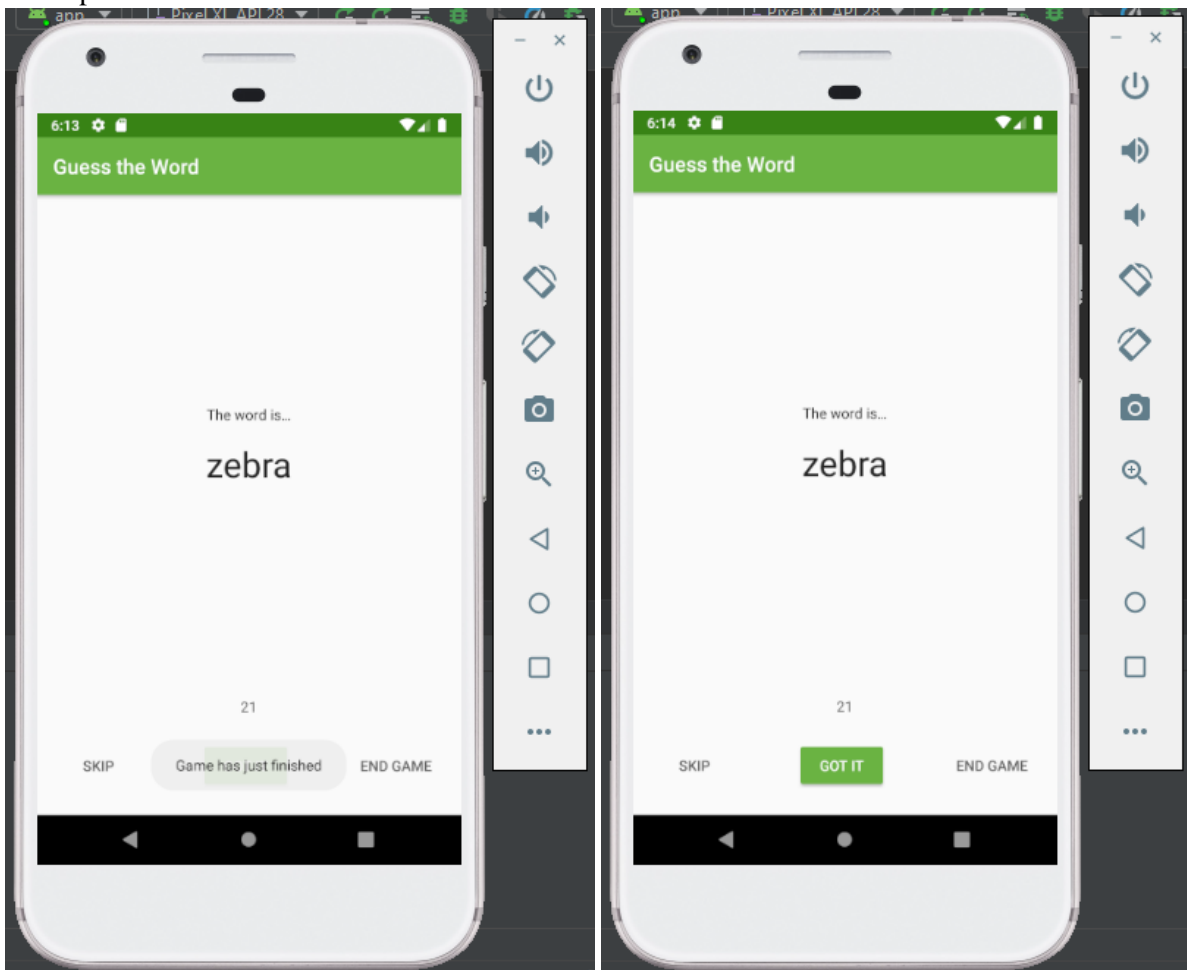
Berpindah ke layar skor tanpa tombol endgame



7. Kode yang Anda tambahkan telah menyebabkan masalah siklus proses. Untuk memahami masalahnya, di kelas `GameFragment`, beri komentar kode navigasi dalam metode `gameFinished()`. Pastikan untuk menyimpan pesan `Toast` dalam metode ini.

```
private fun gameFinished() {  
    Toast.makeText(activity, text: "Game has just finished", Toast.LENGTH_SHORT).show()  
    // val action = GameFragmentDirections.actionGameToScore()  
    // action.score = viewModel.score.value?:0  
    //NavHostFragment.findNavController(this).navigate(action)  
}
```

8. Jalankan aplikasi Anda, mainkan game, dan telusuri semua kata. Pesan bersulang yang mengatakan "Game has just finished" muncul sebentar di bagian bawah layar game, yang merupakan perilaku yang diharapkan.

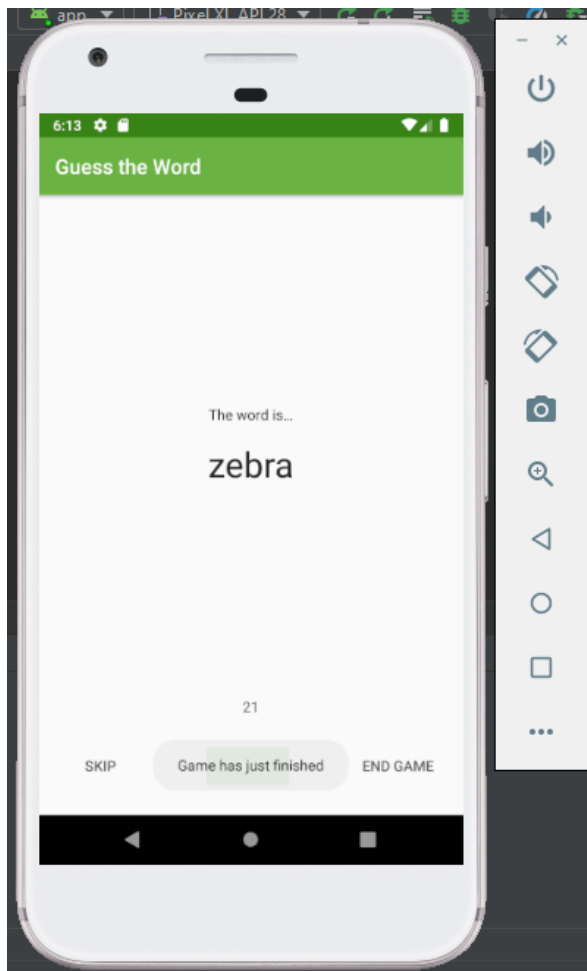


Sekarang putar perangkat atau emulator. `Toast` kembali ditampilkan! Putar perangkat beberapa kali lagi, dan Anda mungkin akan selalu melihat `toast`. Ini adalah bug, karena `toast` hanya ditampilkan sekali, saat game selesai. `Toast` seharusnya tidak ditampilkan setiap kali fragmen dibuat ulang. Anda mengatasi masalah ini di tugas berikutnya.

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)



## Langkah 2: Mereset event game-finished

Biasanya, LiveData mengirimkan update ke pengamat hanya jika data berubah. Pengecualian untuk perilaku ini adalah bahwa pengamat juga menerima pembaruan ketika pengamat berubah dari keadaan tidak aktif menjadi aktif.

Inilah mengapa toast yang diakhiri dengan game dipicu berulang kali di aplikasi Anda. Saat fragmen game dibuat kembali setelah rotasi layar, ia berpindah dari status tidak aktif ke status aktif. Pengamat dalam fragmen dihubungkan kembali ke ViewModel yang ada dan menerima data saat ini. Metode `gameFinished()` dipicu kembali, dan toast ditampilkan.

Dalam tugas ini, Anda memperbaiki masalah ini dan menampilkan toast hanya sekali, dengan menyetel ulang flag eventGameFinish di GameViewModel.

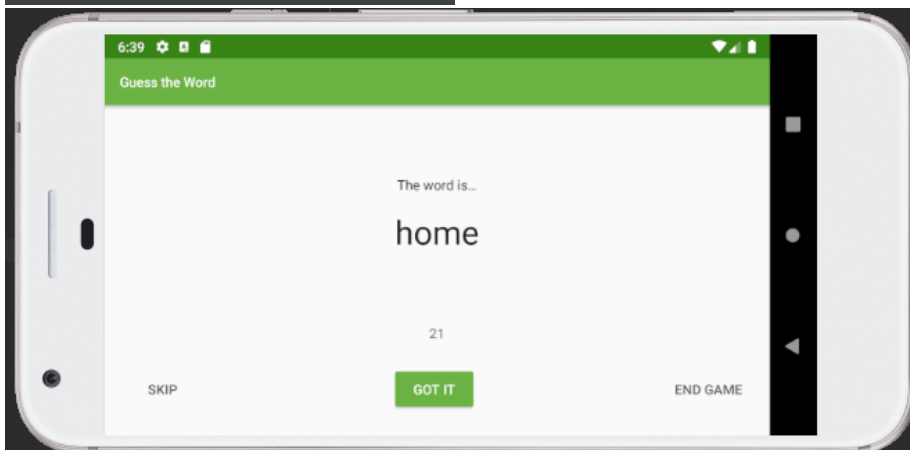
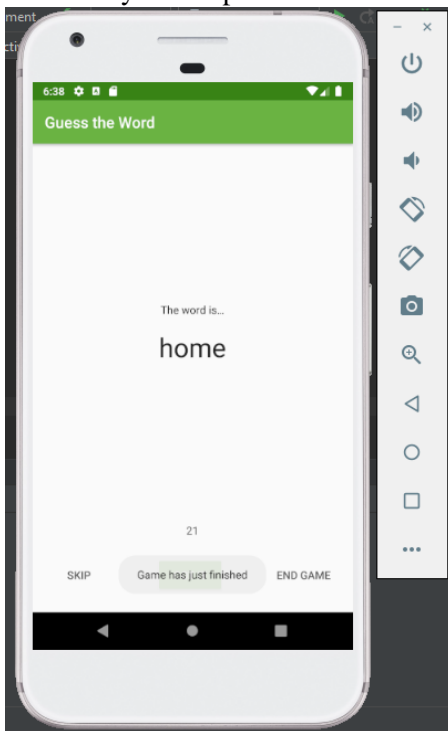
1. Di GameViewModel, tambahkan metode onGameFinishComplete () untuk menyetel ulang event game selesai, \_eventGameFinish.

```
/** Method for the game completed event */  
fun onGameFinishComplete() {  
    _eventGameFinish.value = false  
}
```

2. Dalam GameFragment, di akhir gameFinished(), panggil onGameFinishComplete () pada objek viewModel. (Biarkan kode navigasi di gameFinished () beri komentar untuk saat ini.)

```
private fun gameFinished() {  
    Toast.makeText(activity, text: "Game has just finished", Toast.LENGTH_SHORT).show()  
    // val action = GameFragmentDirections.actionGameToScore()  
    // action.score = viewModel.score.value?:0  
    //NavHostFragment.findNavController(this).navigate(action)  
    viewModel.onGameFinishComplete()  
}
```

3. Jalankan aplikasi dan mainkan gamenya. Telusuri semua kata, lalu ubah orientasi layar perangkat. Toast hanya ditampilkan sekali.



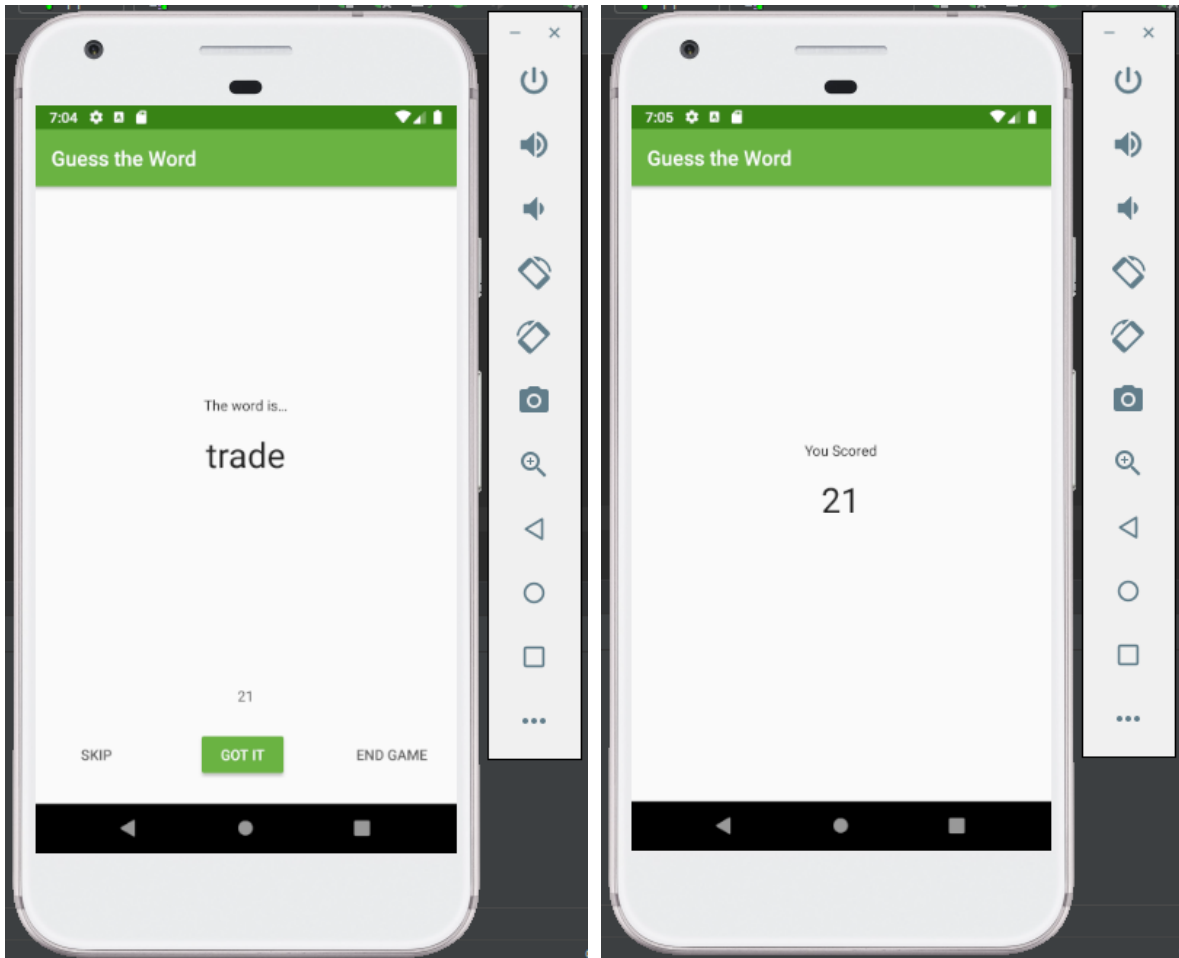


4. Di GameFragment, di dalam metode gameFinished (), hapus tanda komentar pada kode navigasi. Untuk menghapus komentar di Android Studio, pilih baris yang diberi komentar dan tekan Control + /.

```
private fun gameFinished() {  
    Toast.makeText(activity, text: "Game has just finished", Toast.LENGTH_SHORT).show()  
    val action = GameFragmentDirections.actionGameToScore()  
    action.score = viewModel.score.value?:0  
    NavHostFragment.findNavController( fragment: this).navigate(action)  
    viewModel.onGameFinishComplete()  
}
```

Jika diminta oleh Android Studio, impor  
androidx.navigation.fragment.NavHostFragment.findNavController.

5. Jalankan aplikasi dan mainkan gamenya. Pastikan aplikasi menavigasi secara otomatis ke layar skor akhir setelah Anda membaca semua kata.



---

## 8. Tugas: Menambahkan LiveData ke ScoreViewModel

Dalam tugas ini, Anda mengubah skor menjadi objek LiveData di ScoreViewModel dan memasang pengamat padanya. Tugas ini mirip dengan yang Anda lakukan saat menambahkan LiveData ke GameViewModel. Anda membuat perubahan ini pada ScoreViewModel untuk kelengkapan, sehingga semua data di aplikasi Anda menggunakan LiveData.

1. Di ScoreViewModel, ubah jenis variabel score menjadi MutableLiveData. Ubah namanya dengan konvensi menjadi \_score dan tambahkan properti dukungan.

```
// The final score
private val _score = MutableLiveData<Int>()
val score: LiveData<Int>
    get() = _score
```

2. Di ScoreViewModel, di dalam blok init, inialisasi \_score. Anda dapat menghapus atau meninggalkan log di blok init sesuka Anda.

```
init {
    _score.value = finalScore
}
```

3. Di ScoreFragment, di dalam onCreateView (), setelah menginisialisasi viewModel, lampirkan pengamat untuk objek LiveData score. Di dalam ekspresi lambda, setel nilai score ke tampilan teks score. Hapus kode yang secara langsung menetapkan tampilan teks dengan nilai score dari ViewModel.

Kode untuk ditambahkan:

```
// Add observer for score
viewModel.score.observe(viewLifecycleOwner, Observer { newScore ->
    binding.scoreText.text = newScore.toString()
})
```

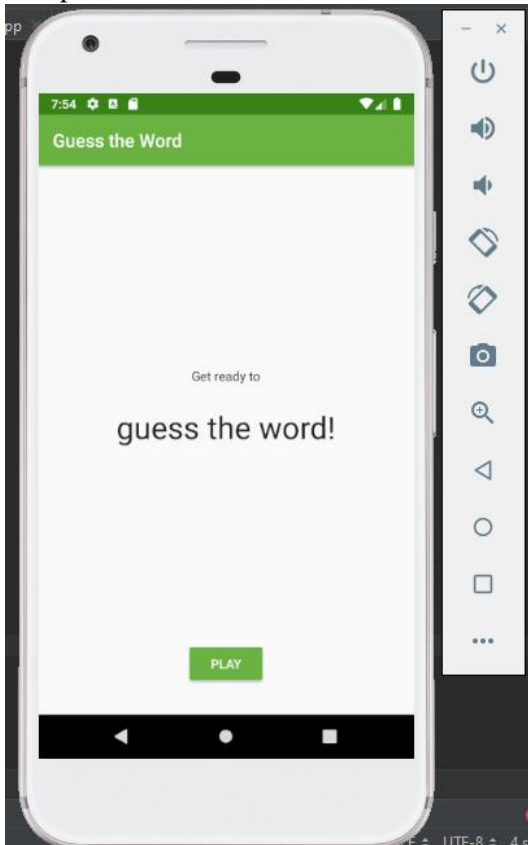
Kode untuk dihapus:

```
binding.scoreText.text = viewModel.score.toString()
```

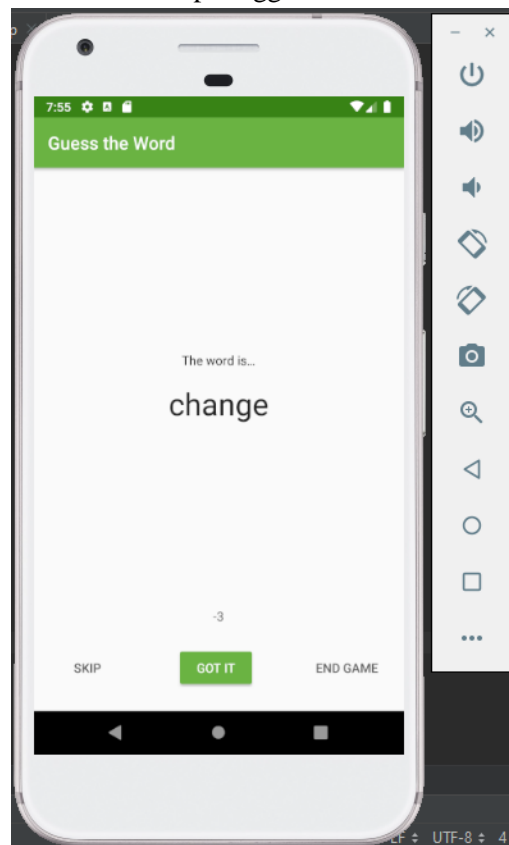
Saat diminta oleh Android Studio, impor androidx.lifecycle.Observer.

4. Jalankan aplikasi Anda dan mainkan gamenya. Aplikasi seharusnya berfungsi seperti sebelumnya, tetapi sekarang menggunakan LiveData dan pengamat untuk memperbarui skor.

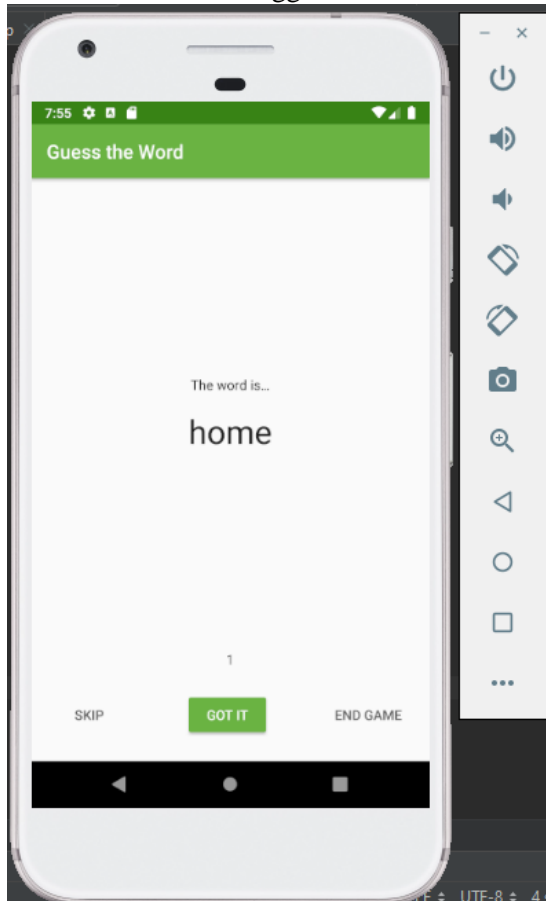
Tampilan awal



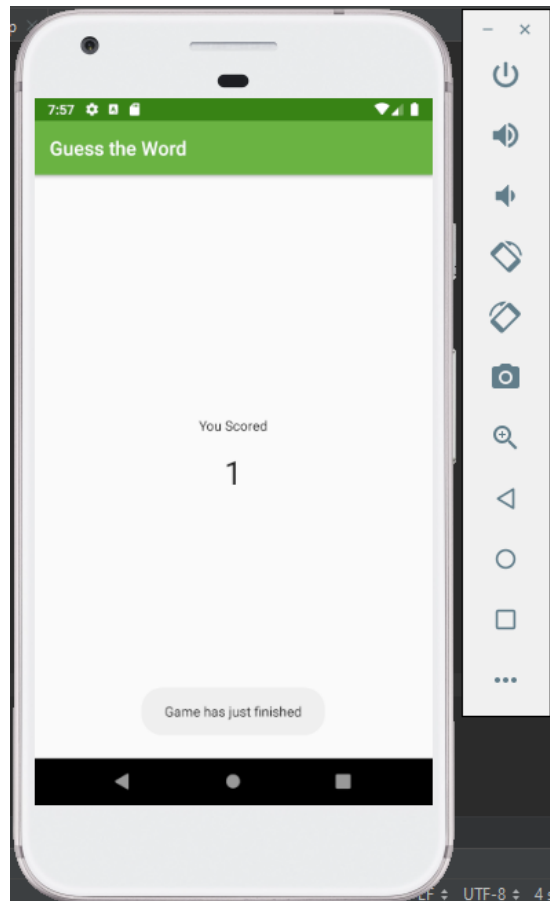
Klik tombol skip hingga -3



Klik tombol Got It hingga 1



Game selesai



---

## 9. Tugas: Tambahkan tombol Putar Lagi

Dalam tugas ini, Anda menambahkan tombol Play Again ke layar skor dan mengimplementasikan pemroses kliknya menggunakan event LiveData. Tombol tersebut memicu event (kejadian) untuk menavigasi dari layar skor ke layar game. Kode awal untuk aplikasi mencakup tombol Play Again, tetapi tombolnya tersembunyi.

1. Di `res/layout/score_fragment.xml`, untuk tombol `play_again_button`, ubah nilai atribut `visibility` menjadi `visible`.

```
<Button
    android:id="@+id/play_again_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:text="Play Again"
    android:theme="@style/GoButton"
    android:visibility="visible"
```

2. Di `ScoreViewModel`, tambahkan objek LiveData untuk memegang Boolean yang disebut `_eventPlayAgain`. Objek ini digunakan untuk menyimpan acara LiveData untuk menavigasi dari layar skor ke layar game.

```
private val _eventPlayAgain = MutableLiveData<Boolean>()
val eventPlayAgain: LiveData<Boolean>
    get() = _eventPlayAgain
```

3. Di `ScoreViewModel`, tentukan metode untuk menyetel dan menyetel ulang acara, `_eventPlayAgain`.

```

fun onPlayAgain() {
    _eventPlayAgain.value = true
}
fun onPlayAgainComplete() {
    _eventPlayAgain.value = false
}

```

4. Di ScoreFragment, tambahkan pengamat untuk eventPlayAgain. Letakkan kode di akhir onCreateView (), sebelum pernyataan return. Di dalam ekspresi lambda, navigasikan kembali ke layar game dan setel ulang eventPlayAgain.

```

// Navigates back to game when button is pressed
viewModel.eventPlayAgain.observe(viewLifecycleOwner, Observer { playAgain ->
    if (playAgain) {
        findNavController().navigate(ScoreFragmentDirections.actionRestart())
        viewModel.onPlayAgainComplete()
    }
})

```

Impor androidx.navigation.fragment.findNavController, jika diminta oleh Android Studio.

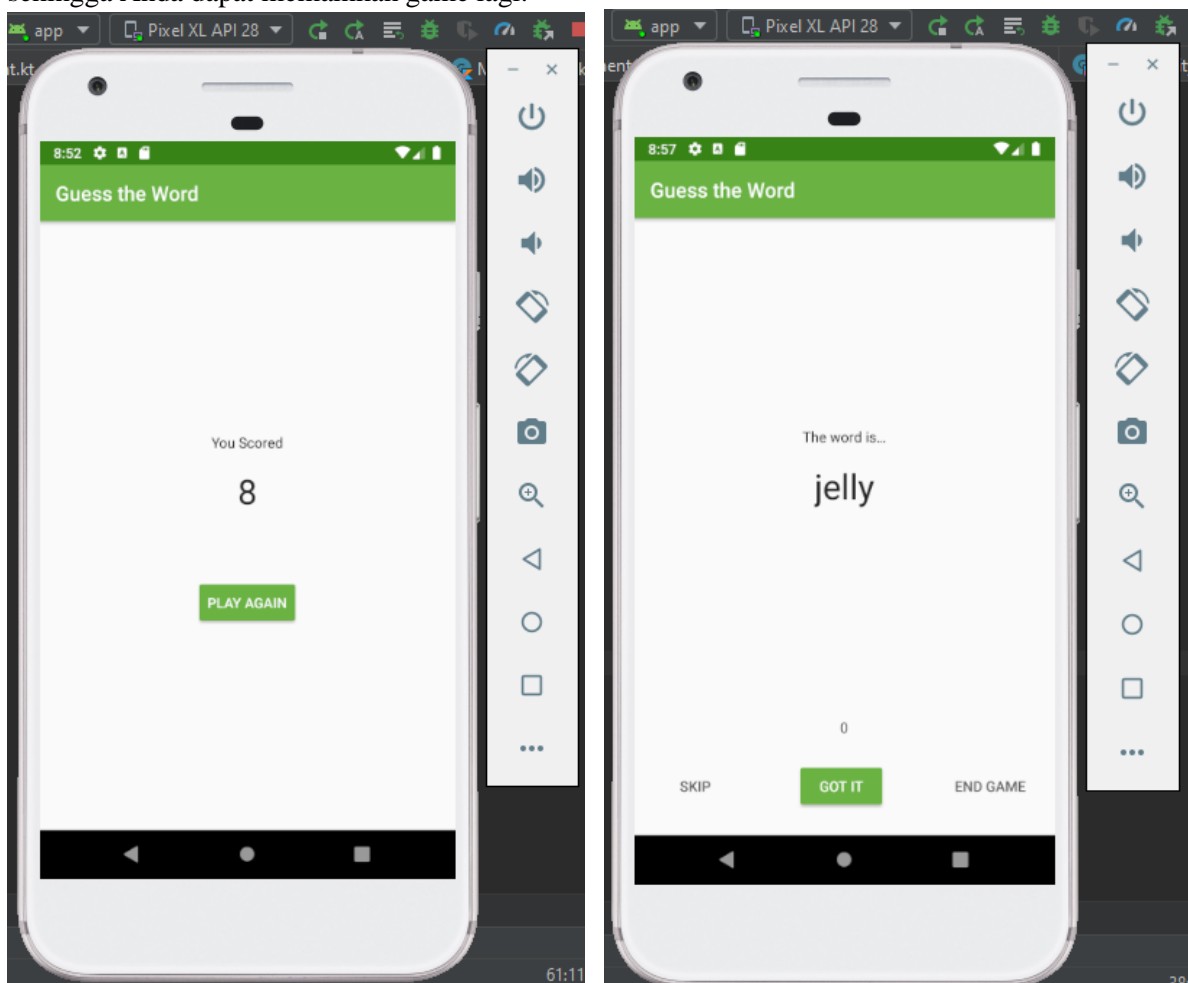
5. Di ScoreFragment, di dalam onCreateView (), tambahkan listener klik ke tombol PlayAgain dan panggil viewModel.onPlayAgain ().

```

binding.playAgainButton.setOnClickListener { viewModel.onPlayAgain() }
return binding.root

```

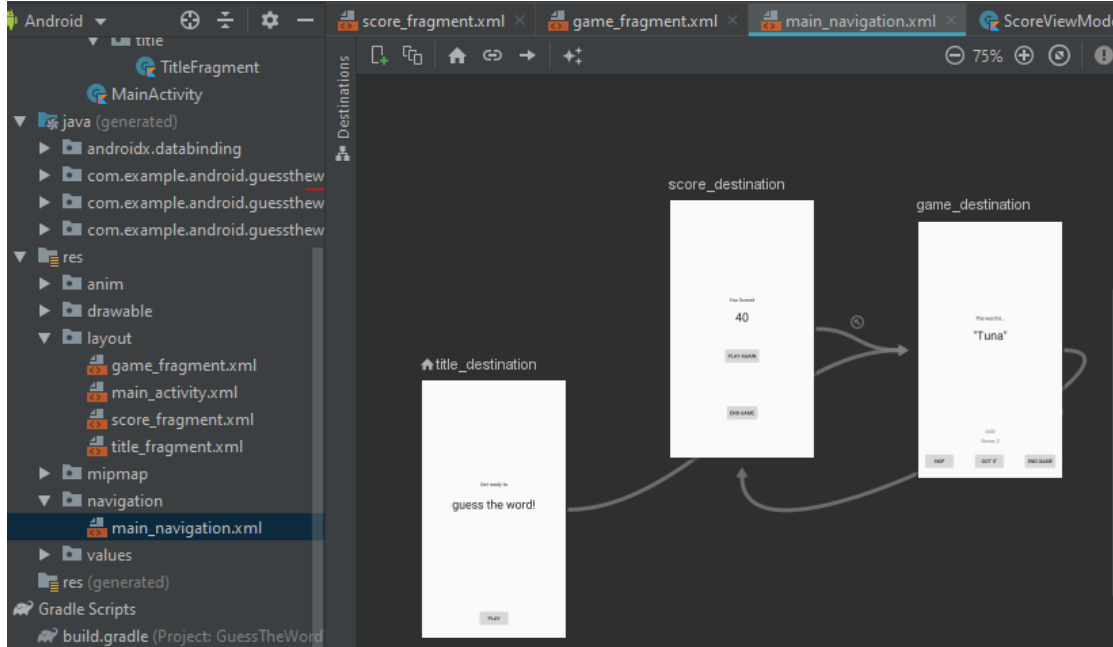
6. Jalankan aplikasi Anda dan mainkan gamenya. Saat permainan selesai, layar skor menunjukkan skor akhir dan tombol Play Again. Ketuk tombol Play Again, dan aplikasi menavigasi ke layar game sehingga Anda dapat memainkan game lagi.



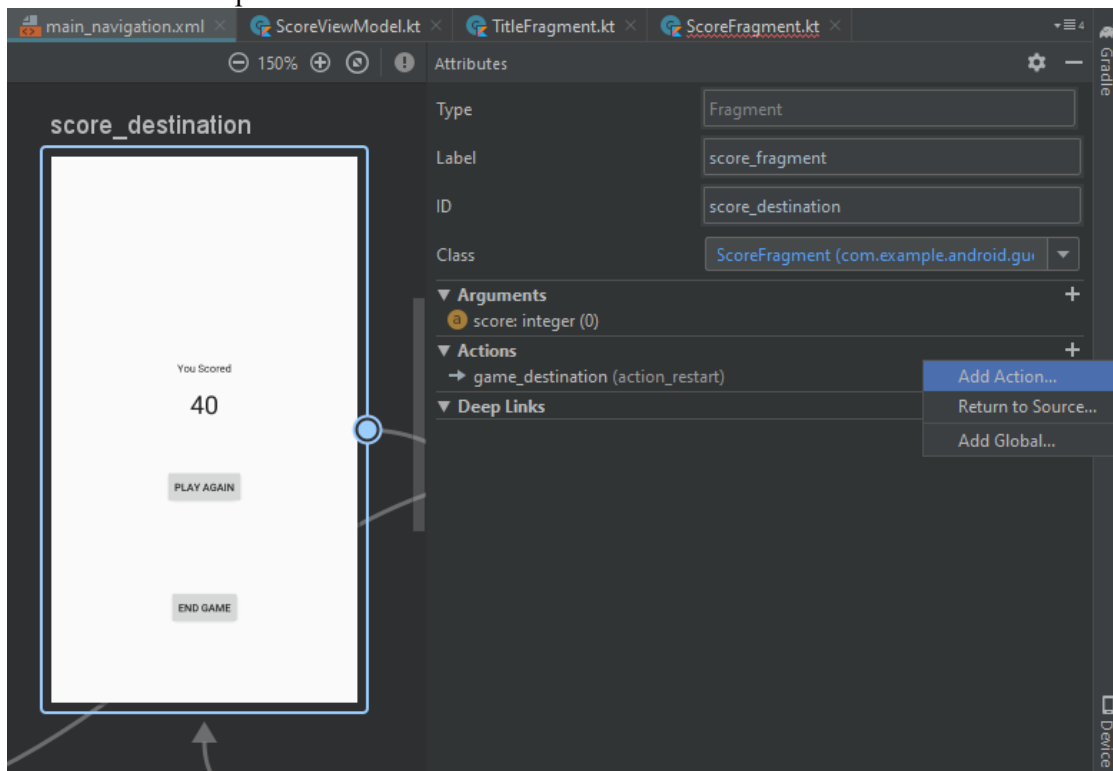
## LATIHAN

Di `res/layout/score_fragment.xml` tambahkan tombol End Game yang akan menavigasi ke layar judul.

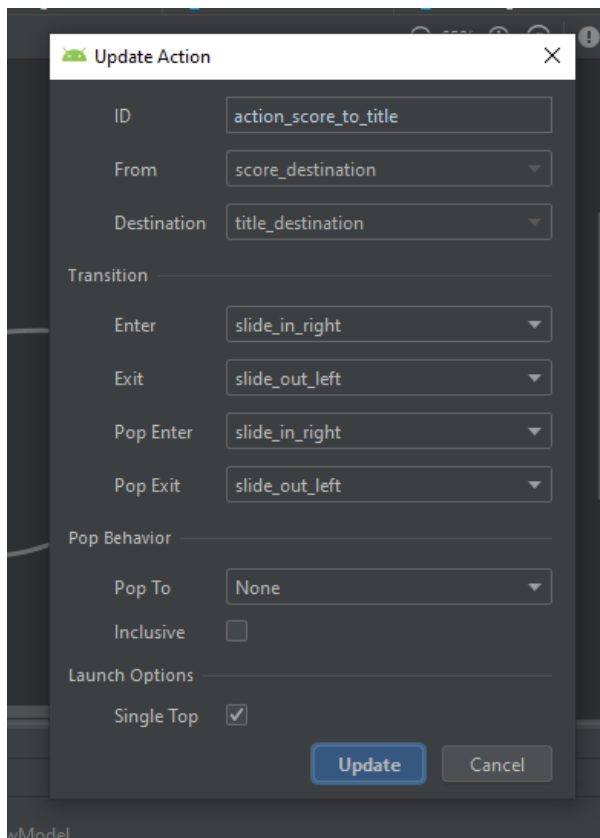
1. Masuk ke `main_navigation.xml`



2. Pilih `score_destination`, kemudian di toolbar Attributes langsung saja klik tombol “+” di samping kanan Actions lalu pilih “Add Action..”



3. Di Update Action, pilih konfigurasi seperti gambar berikut, kemudian klik Add



4. Sehingga di tab text kodenya adalah seperti berikut

```
<action
    android:id="@+id/action_score_to_title"
    app:destination="@id/title_destination"
    app:enterAnim="@anim/slide_in_right"
    app:exitAnim="@anim/slide_out_left"
    app:popEnterAnim="@anim/slide_in_right"
    app:popExitAnim="@anim/slide_out_left"
    app:launchSingleTop="true" />
</fragment>
```

5. Pindah ke ScoreViewModel, lalu tambahkan objek LiveData untuk meng-handle Boolean yang disebut `_eventEndGame`. Objek ini digunakan untuk menyimpan event LiveData untuk menavigasi dari layar skor ke layar title.

```
private val _eventEndGame = MutableLiveData<Boolean>()
val eventEndGame: LiveData<Boolean>
    get() = _eventEndGame
```

6. Buat method untuk menyetel ulang event, `_eventEndGame` bernilai boolean true dan false

```
fun onEndGame() {
    _eventEndGame.value = true
}
fun onEndGameComplete() {
    _eventEndGame.value = false
}
```

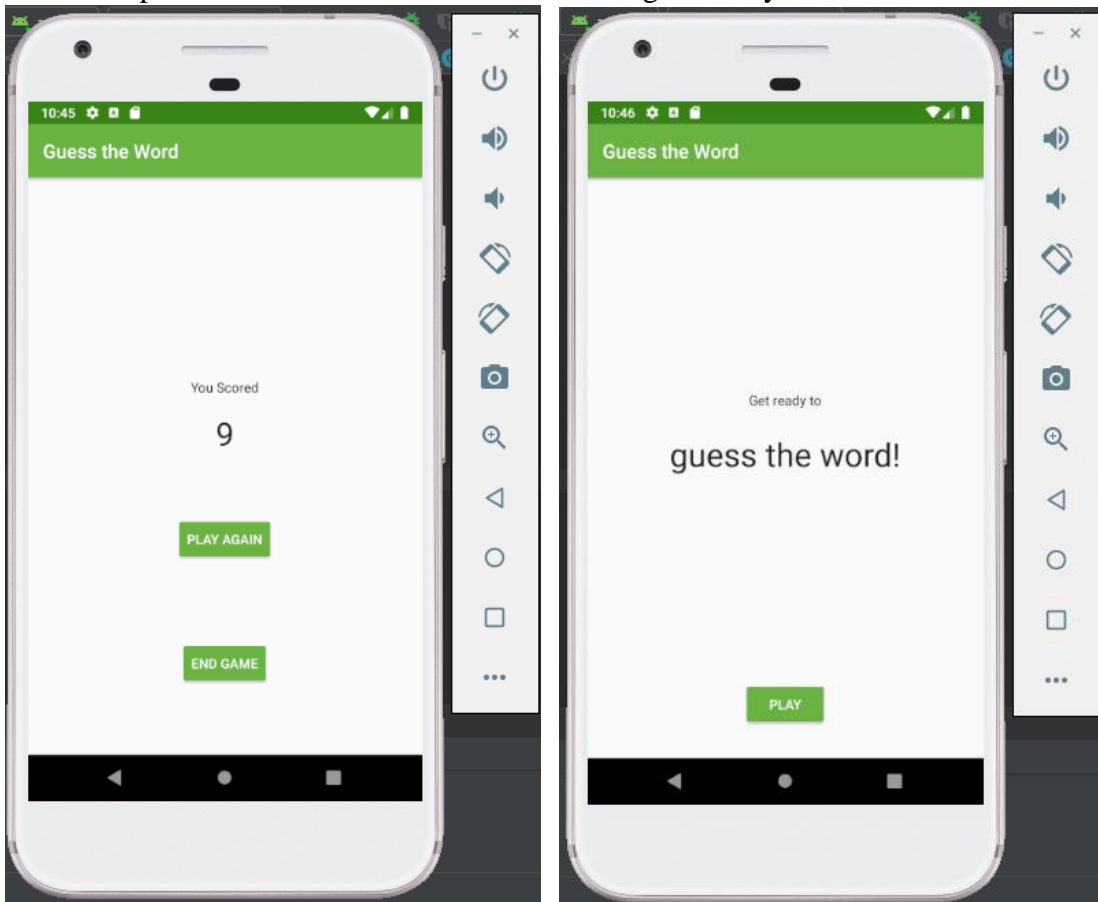
7. Pindah ke ScoreFragment, kemudian tambahkan viewModel untuk `eventEndGame`. Letakkan kode di akhir `onCreateView()`, sebelum pernyataan `return`. Di dalam ekspresi lambda, navigasikan kembali ke layar title dan setel ulang `eventEndGame`.

```
viewModel.eventEndGame.observe(viewLifecycleOwner, Observer { endGame ->
    if (endGame) {
        findNavController().navigate(ScoreFragmentDirections.actionScoreToTitle())
        viewModel.onEndGameComplete()
    }
})
```

- Di dalam onCreateView (), tambahkan listener klik ke tombol EndGame dan panggil viewModel.onEndGame().

```
binding.endGameButton.setOnClickListener { viewModel.onEndGame() }
```

- Jalankan aplikasi. Button End Game akan menavigasi ke layar title



## TUGAS

Buat aplikasi baru dengan menerapkan LiveData dan LiveData observer.

- Buat project baru bernama LiveDataTugas

Name

Package name

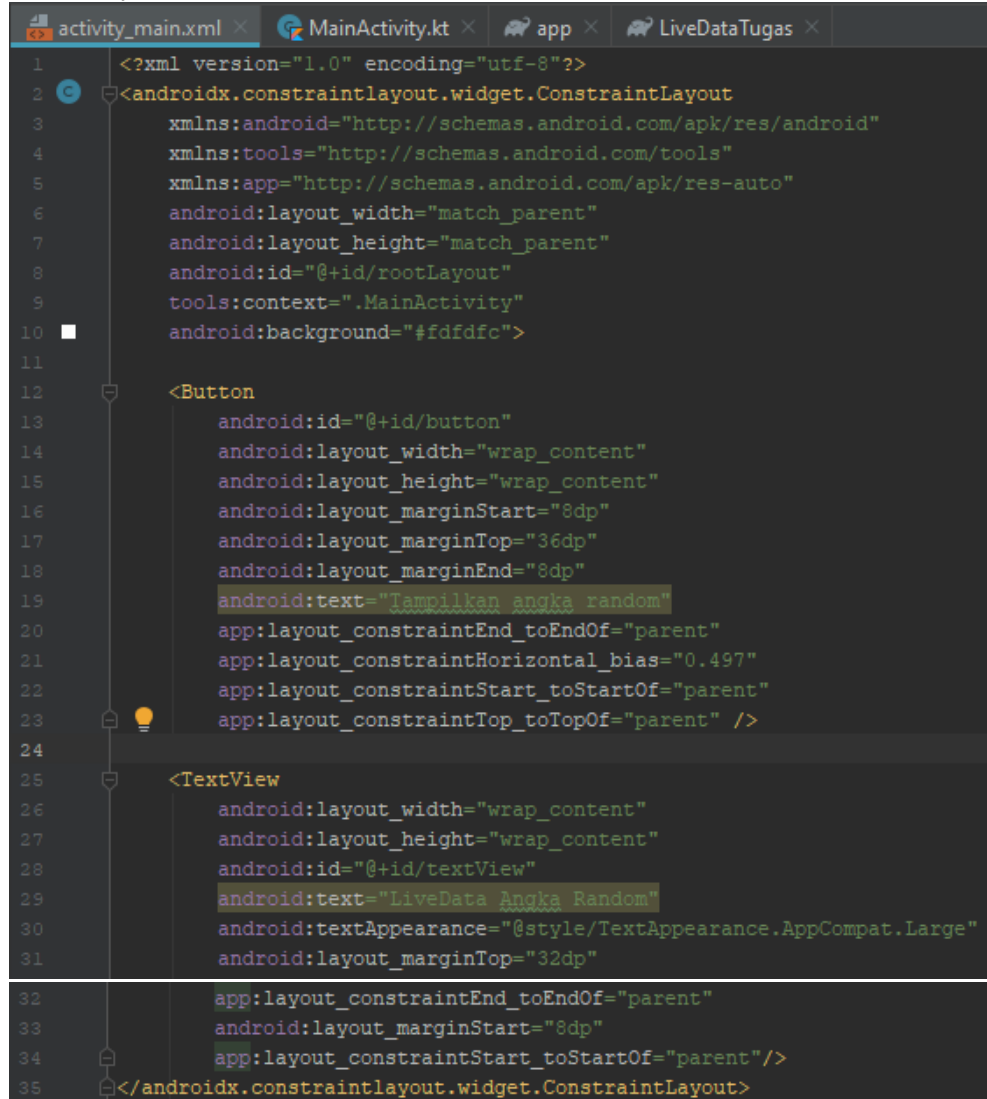
2. Di build.gradle (Module: app), tambahkan dependensi berikut

```
implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
```

Penjelasan:

Di dependensi ini kita menggunakan lifecycle. Apa itu lifecycle? Lifecycle adalah kelas/interface yang menyimpan informasi tentang status aktivitas/fragmen dan juga memungkinkan objek lain untuk mengamati status ini dengan melacak atau memantau status yang bersangkutan tersebut.

3. Di activity\_main.xml, tulis kode berikut:



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/rootLayout"
    tools:context=".MainActivity"
    android:background="#fdddfc">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="36dp"
        android:layout_marginEnd="8dp"
        android:text="Tampilkan angka random"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView"
        android:text="LiveData Angka Random"
        android:textAppearance="@style/TextAppearance.AppCompat.Large"
        android:layout_marginTop="32dp"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginStart="8dp"
        app:layout_constraintStart_toStartOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Penjelasan:

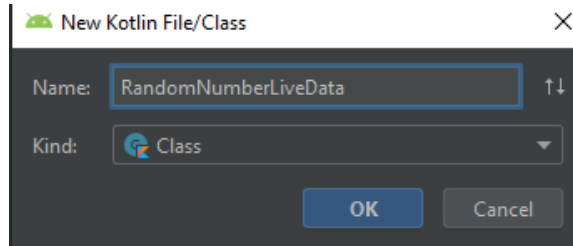
Di xml ini, kita menggunakan ConstraintLayout merupakan salah satu komponen ViewGroup yang dapat kita gunakan untuk menyusun tampilan aplikasi yang kompleks tanpa adanya nested layout. ConstraintLayout tersedia dengan dukungan kompatibilitas mulai dari Android 2.3 (API Level 9) sampai dengan yang terbaru. Layout ini berbasis relative layout, namun mempunyai tingkat kemudahan yang lebih baik dalam penggunaannya. Hal ini dikarenakan, constraint layout dapat digunakan dengan baik pada design mode didalam Android Studio. Setiap item pada constraint layout memiliki 4 arah constraint yaitu top, left, right, dan bottom. Ke empat arah ini memiliki sebuah connection source yang dapat ditarik ke parent atau ke objek lain. Kita bisa menggunakan sistem drag and drop dalam menerapkan constraint layout.

Di dalam constraintlayout tersebut, kita buat dua widget yaitu sebuah button dan sebuah textview. Button yang kita buat memiliki id bernama "button" dan menampilkan pesan "Tampilkan angka random". Button ini saat diklik akan menampilkan angka random pada textview di bawahnya.



Sementara textview yang kita buat menampilkan pesan “LiveData Angka Random” dan juga menampilkan angka random saat nanti button sudah diklik.

4. Buat kelas baru bernama RandomNumberLiveData



5. Di dalam class RandomNumberLiveData tersebut, tulis kode berikut:

```
1 package com.example.livedatatugas
2
3 import androidx.lifecycle.MutableLiveData
4 import androidx.lifecycle.ViewModel
5
6 class RandomNumberLiveData: ViewModel() {
7     val currentRandomNumber: MutableLiveData<Int> by lazy {
8         MutableLiveData<Int>()
9     }
10 }
```

Penjelasan:

tambahkan class RandomNumberLiveData untuk LiveData yang berfungsi untuk meng-handle integer untuk number yang disebut currentRandomNumber. Variable currentRandomNumber digunakan untuk menyimpan event LiveData.

6. Di dalam MainActivity, tulis kode berikut:

```
1 package com.example.livedatatugas
2
3 import androidx.appcompat.app.AppCompatActivity
4 import android.os.Bundle
5 import androidx.lifecycle.ViewModelProvider
6 import androidx.lifecycle.Observer
7 import kotlinx.android.synthetic.main.activity_main.*
8 import java.util.*
9
10 class MainActivity : AppCompatActivity() {
11
12     private lateinit var mModel : RandomNumberLiveData
13
14     override fun onCreate(savedInstanceState: Bundle?) {
15         super.onCreate(savedInstanceState)
16         setContentView(R.layout.activity_main)
17
18         mModel = ViewModelProvider( owner: this).get(RandomNumberLiveData::class.java)
19
20         val randomNumberObserver = Observer<Int> { newNumber ->
21             textView.text = "Angka random baru : $newNumber"
22         }
23         mModel.currentRandomNumber.observe( owner: this, randomNumberObserver)
24
25         button.setOnClickListener { it: View!
26
27             mModel.currentRandomNumber.value = Random().nextInt( bound: 50)
28         }
29     }
30 }
```

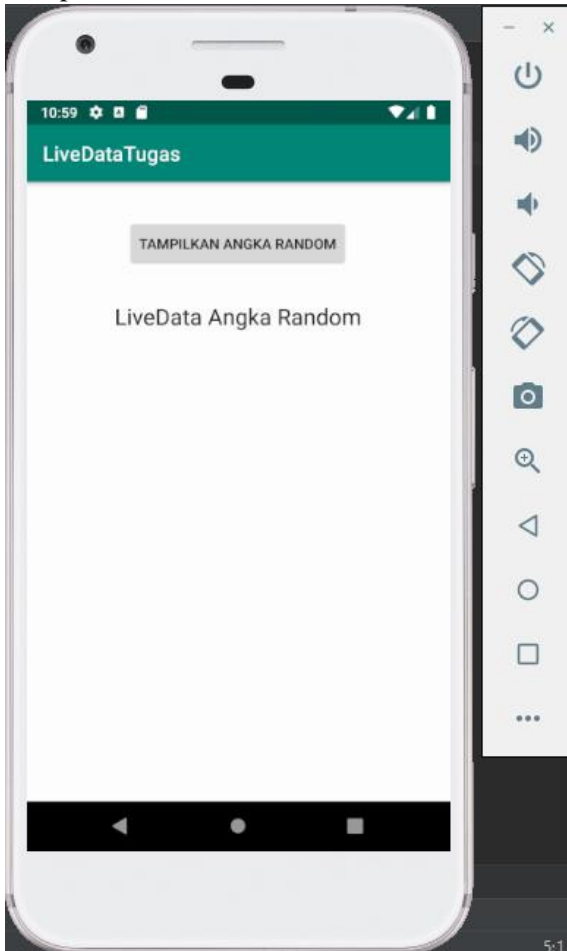
Penjelasan:

Deklarasikan variabel `mModel` untuk meng-handle event di class `RandomNumberLiveData`, kemudian di dalam method `onCreate`, gunakan `mModel` untuk menggunakan `ViewModelProvider` untuk mendapatkan event `ViewModel` dari `RandomNumberLiveData`. Untuk melakukan itu kita tulis **`mModel = ViewModelProvider(this).get(RandomNumberLiveData::class.java)`**. Kemudian kode yang kita tulis adalah **`val randomNumberObserver = Observer<Int>{newNumber->`** yang berfungsi untuk membuat observer yang meng-update UI dan **`textView.text = "Angka random baru : $newNumber"`** yang berfungsi untuk meng-update UI dengan data baru.

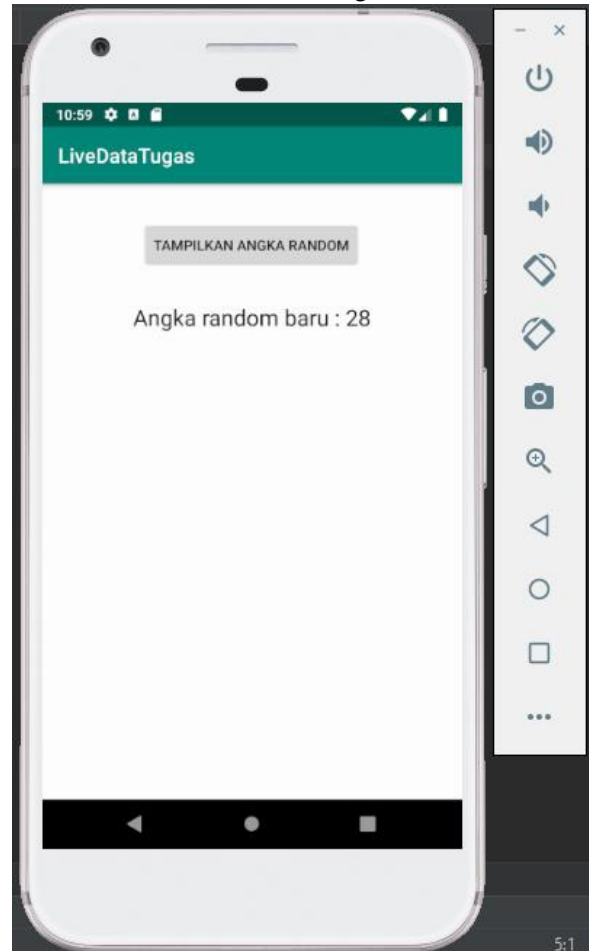
Selanjutnya adalah **`mModel.currentRandomNumber.observe(this,randomNumberObserver)`** yang berguna untuk meng-observe livedata dan membuat activity sebagai owner lifecycle dan observer. Dilanjutkan dengan **`button.setOnClickListener`** yang digunakan untuk listener button yang tadi sudah kita buat supaya nanti muncul event saat button tersebut diklik. Terakhir, kita ubah data yang ada di dalam aplikasi dengan **`mModel.currentRandomNumber.value = Random().nextInt(50)`** yang berarti program akan memilih angka random antara 0-50.

7. Jalankan aplikasi di emulator Pixel XL API 28

Tampilan awal



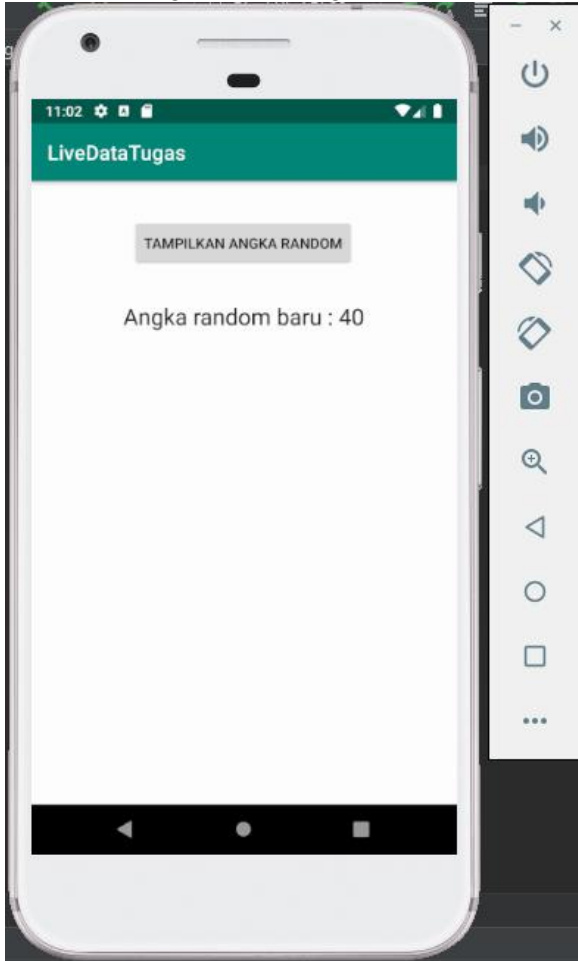
saat button diklik, muncul angka random 28



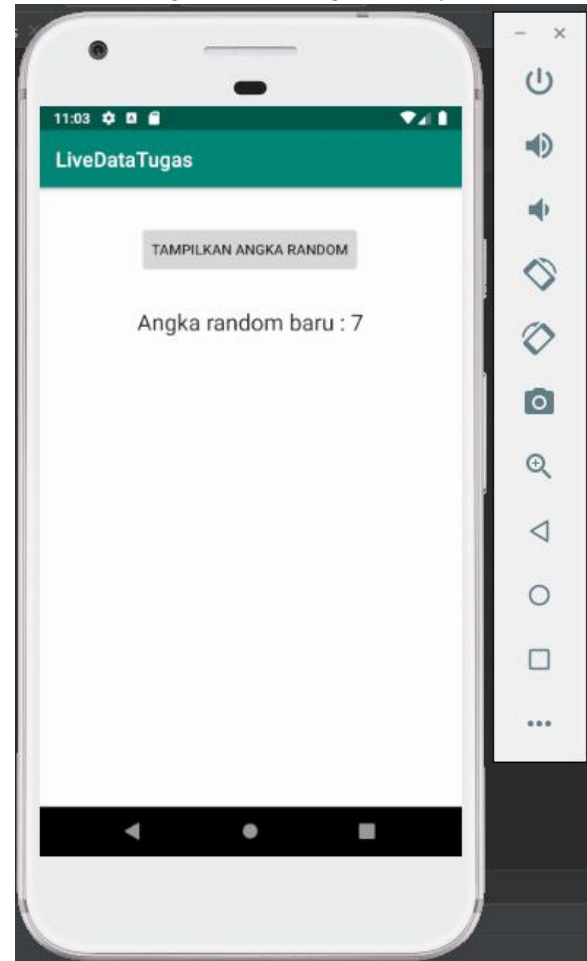
(screenshot lain di halaman selanjutnya)

(screenshot lain di halaman selanjutnya)

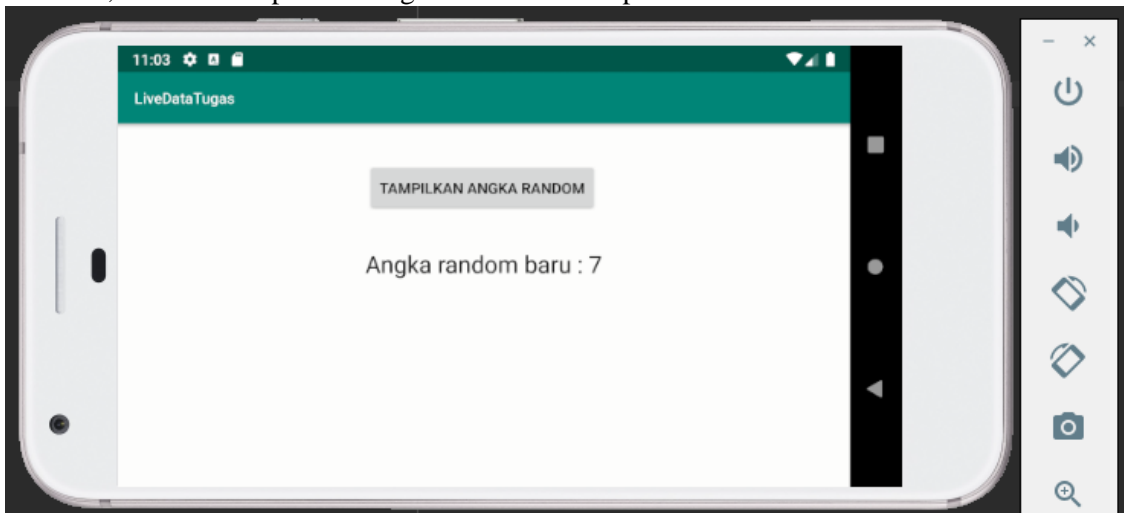
Klik button lagi, muncul angka random baru



klik button lagi, muncul angka baru yaitu 7



Terakhir, coba rotate aplikasi. Angka 7 masih tersimpan.



---

## KESIMPULAN

Pada pertemuan ke-9 ini, saya berhasil memenuhi tujuan dibuatnya modul ini yaitu mampu menggunakan LiveData dan LiveData observer dalam aplikasi. Pada pembuatan laporan ini dari mengerjakan praktik, latihan, hingga tugas, saya mempelajari banyak hal seperti Bagaimana menambahkan LiveData ke data yang disimpan

di ViewModel, Kapan dan bagaimana menggunakan MutableLiveData, Bagaimana menambahkan metode observer untuk mengamati perubahan di LiveData, cara berkomunikasi antara UI controller dan ViewModel, dan lain-lain. Di praktik, saya menggunakan aplikasi starter yang nantinya kita tambahkan beberapa kode untuk menerapkan livedata. Di latihan, saya menambahkan tombol endgame supaya bisa kembali ke layar title. Dan terakhir di tugas, saya menerapkan livedata untuk menampilkan angka random (acak).

Terima Kasih