MODUL 9

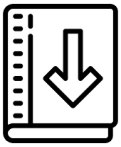
LiveData dan LiveData Observer

CAPAIAN PEMBELAJARAN

1. Mahasiswa mampu menggunakan LiveData dan LiveData observer dalam aplikasi

KEBUTUHAN ALAT/BAHAN/SOFTWARE

1. Android Studio 3.5
2. Handphone Android versi 7.0 (Nougat)
3. Kabel data USB.
4. Driver ADB.



DASAR TEORI

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

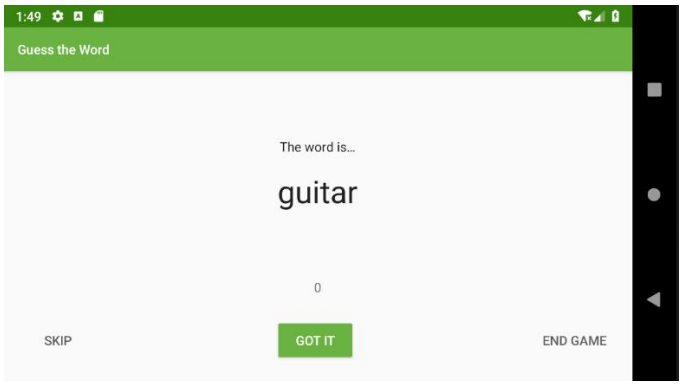
- Gunakan **LiveData** untuk kata dan skor di aplikasi **GuessTheWord**.
- Tambahkan pengamat yang memperhatikan ketika kata atau skor berubah.
- Perbarui tampilan teks yang menampilkan nilai yang diubah.
- Gunakan pola pengamat LiveData untuk menambahkan acara yang sudah selesai game.
- Terapkan tombol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

Pemain pertama melihat kata-kata di aplikasi dan memerankannya secara bergantian, pastikan untuk tidak menampilkan kata tersebut ke pemain kedua. Pemain kedua mencoba menebak kata tersebut.

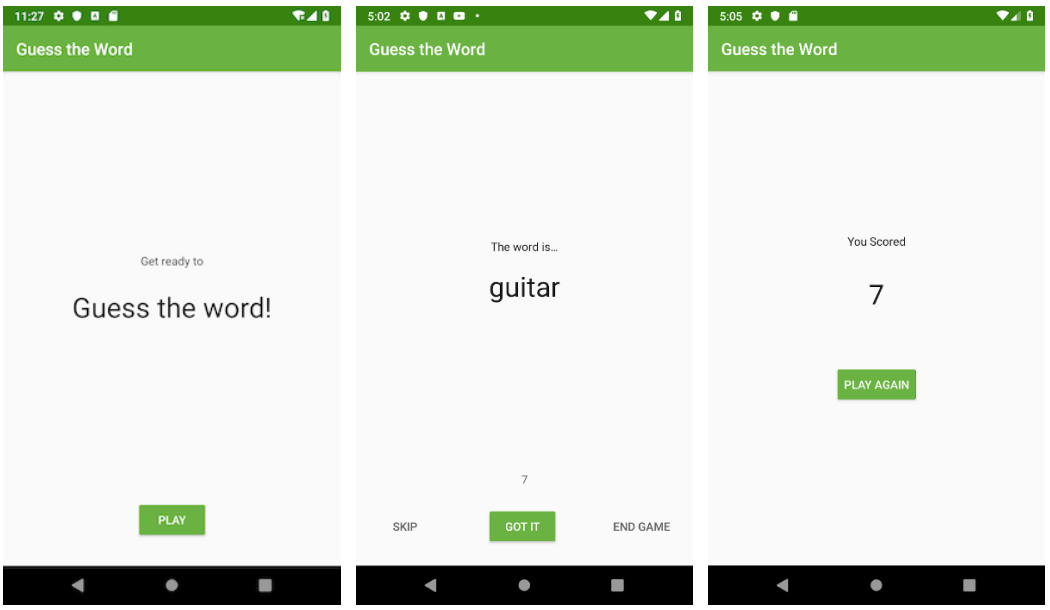
Untuk memainkan game, pemain pertama membuka aplikasi di perangkat dan melihat sebuah kata, misalnya "gitar", seperti yang ditunjukkan pada gambar di bawah



Pemain pertama memerankan kata, berhati-hatilah agar tidak benar-benar mengucapkan kata itu sendiri.

- Saat pemain kedua menebak kata dengan benar, pemain pertama menekan tombol **Got It** (Mengerti), yang menambah hitungan satu dan menampilkan kata berikutnya.
- Jika pemain kedua tidak dapat menebak kata, pemain pertama menekan tombol **Skip** (Lewati), yang akan mengurangi hitungan satu kali dan melompat ke kata berikutnya.
- Untuk mengakhiri permainan, tekan tombol **End Game**. (Fungsi ini tidak ada dalam kode awal untuk codelab pertama dalam seri ini.)

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.



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.
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 Fragmen). 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
  ...
}
```

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(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(1)
   nextWord()
}
```

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

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

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

```
/** Methods for updating the UI **/
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, "Game has just finished", Toast.LENGTH_SHORT).show()
   val action = GameFragmentDirections.actionGameToScore()
   action.score = viewModel.score.value?:0
   NavHostFragment.findNavController(this).navigate(action)
}
```

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

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**.

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

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>
```

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**.

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

5. Di **GameViewModel**, ubah referensi **score** ke versi internalnya yang dapat berubah, _**score**.

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

...
fun onSkip() {
   _score.value = (score.value)?.minus(1)
  ...
}

fun onCorrect() {
   _score.value = (score.value)?.plus(1)
   ...
}
```

6. 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 = ""
   ...
}
...
private fun nextWord() {
   if (!wordList.isEmpty()) {
       //Select and remove a word from the list
       _word.value = wordList.removeAt(0)
   }
}
```

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

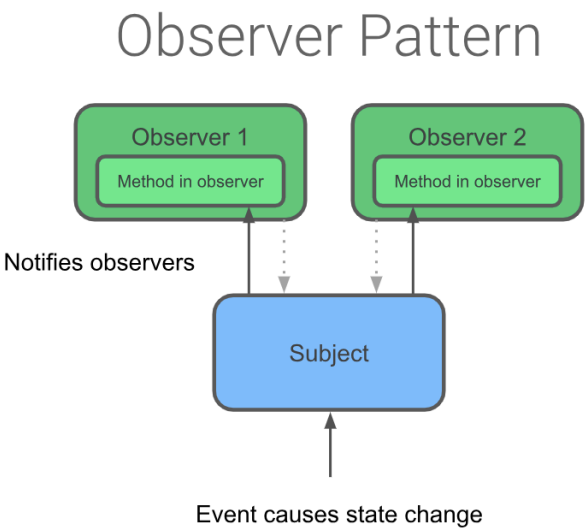
7. Tugas: Menambahkan event game selesai

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 list
       _word.value = wordList.removeAt(0)
   }
}
```

5. Di **GameFragment**, di dalam **onCreateView ()**, setelah menginisialisasi viewModel, lampirkan pengamat ke eventGameFinish. Gunakan metode observ (). 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.

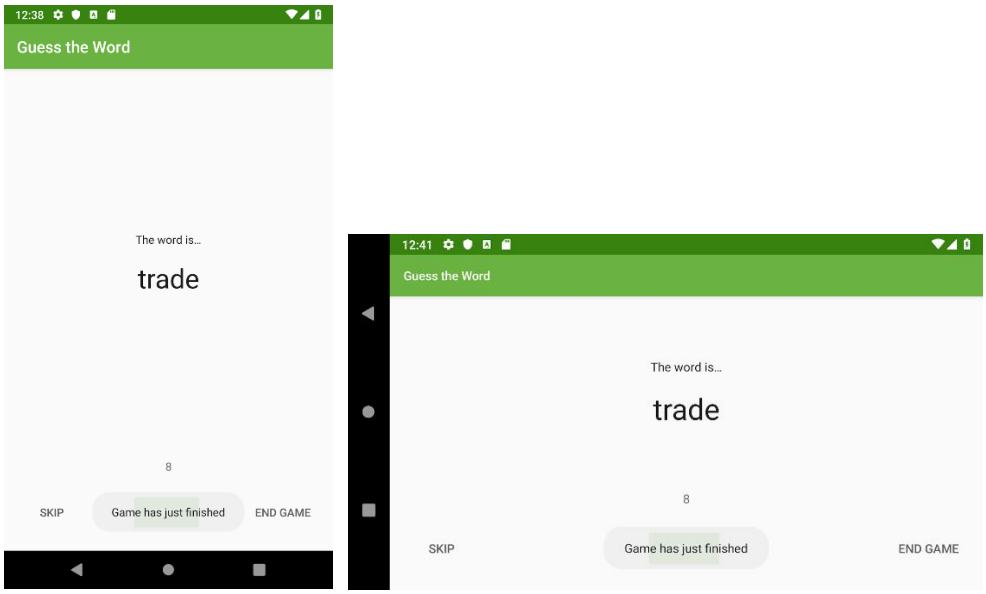
   Setelah daftar kata kosong, **eventGameFinish** disetel, metode pengamat terkait dalam fragmen game dipanggil, dan aplikasi menavigasi ke fragmen layar.

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, "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.



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() {
   ...
   viewModel.onGameFinishComplete()
}
```

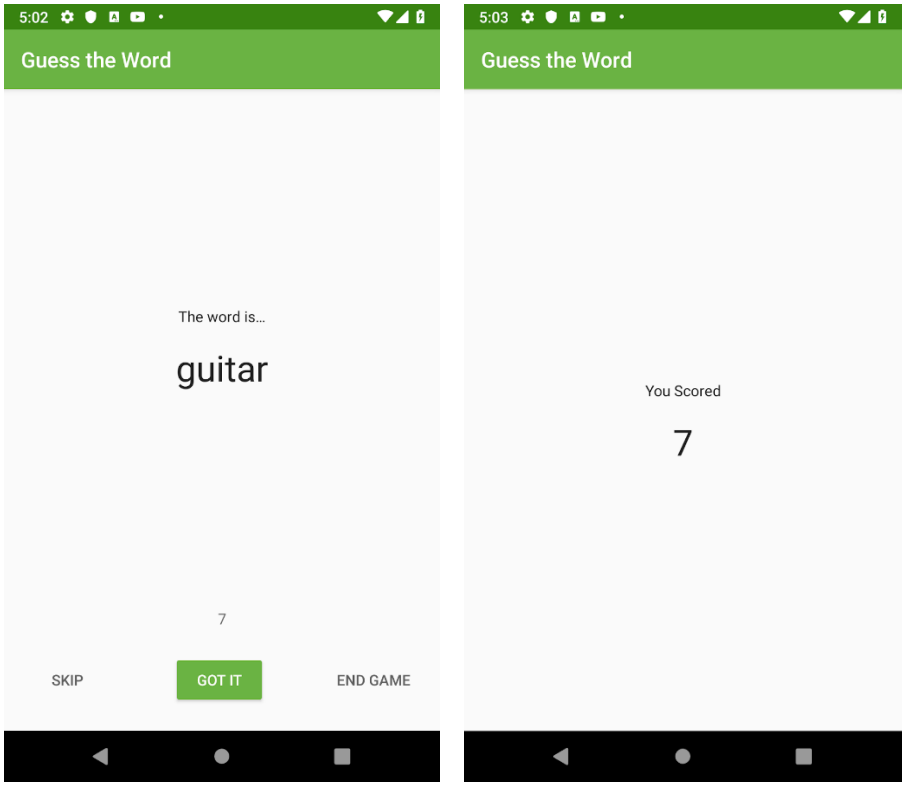
3. Jalankan aplikasi dan mainkan gamenya. Telusuri semua kata, lalu ubah orientasi layar perangkat. Toast hanya ditampilkan sekali.
4. Di **GameFragment**, di dalam metode **gameFinished ()**, hapus tanda komentar pada kode navigasi.

   Untuk menghapus komentar di Android Studio, pilih baris yang diberi komentar dan **tekan Control + /** .

```
private fun gameFinished() {
   Toast.makeText(activity, "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()
}
```

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.

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

2. Di **ScoreViewModel**, di dalam blok **init**, inisialisasi **_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.

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: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**.

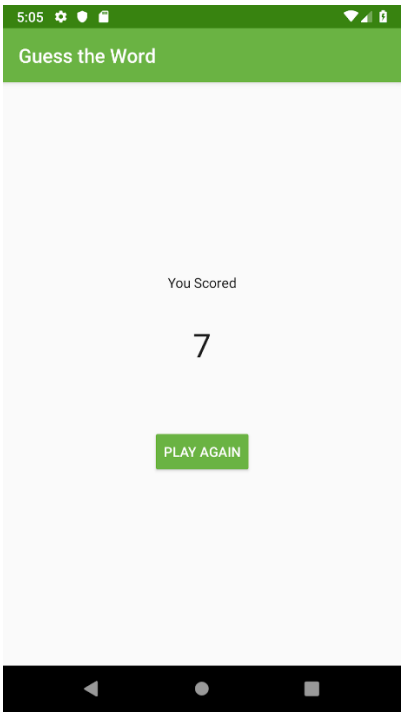
```
// Navigates back to game when button is pressed
viewModel.eventPlayAgain.observe(viewLifecycleOwner, Observer { playAgain ->
   if (playAgain) {
      findNavController().navigate(ScoreFragmentDirections.actionRestart())
       viewModel.onPlayAgainComplete()
   }
})
```

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()  }
```

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

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

TUGAS

1. Buat aplikasi baru dengan menerapkan LiveData dan LiveData observer.

REFERENSI

1. https://developer.android.com/codelabs/kotlin-android-training-live-data#0