Họ tên: Lê Phúc Hưng

MSSV: 20215276

Mã lớp: 151902

Môn học: Phát triển ứng dụng cho thiết bị di động (IT4785)

Source Code: https://github.com/lephuchung/HustMobile

Bài: Lesson 12 – Store data in ViewModel – Use LiveData with ViewModel

Contents

1. Store data in ViewModel

1.1. Before you begin

Tải file mã nguồn mẫu trên github

1.2. Start app overview

Game unscramble app là một trò chơi sắp xếp lại các chữ cái để có được một từ có nghĩa trong từ điển.

Ta sẽ tải file nguồn mẫu về từ github và thay đổi trên đó

1.3. Learn about app architecture

1.4. Add a ViewModel

Trong phần này ta sẽ add một ViewModel giúp ta chứa các dữ liệu của ứng dụng.

Toàn bộ app sẽ được thiết kế theo cấu trúc như sau:

MainActivity → GameFragment → Game ViewModel. GameFragment được gọi để hiển thị từ MainActivity và sẽ lấy thông tin dữ liệu từ Game ViewModel

Để sử dụng được ViewModel ta cần thêm dependencies

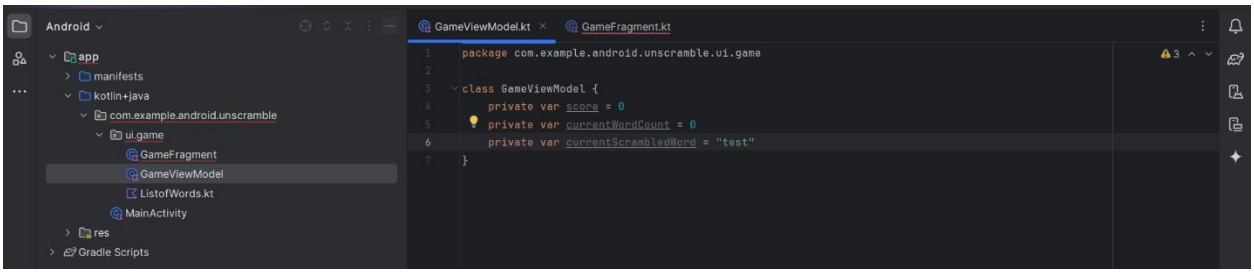
```
// ViewModel
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.5.1'
```

Ở đầu lớp GameFragment, thêm một thuộc tính thuộc loại GameViewModel.

Khởi tạo GameViewModel bằng cách sử dụng tính năng uỷ quyền thuộc tính by viewModels() của Kotlin.

1.5. Move data to ViewModel

Ta di chuyển các biến sang lớp GameViewModel



Ta tiếp thục thêm thuộc tính sao lưu vào currentScrambleWord:

```
GameViewModel.kt × GameFragment.kt
1  package com.example.android.unscramble.ui.game
2
3  class GameViewModel {
4      private var score = 0
5      private var currentWordCount = 0
6      private var _currentScrambledWord = "test"
7      val currentScrambledWord: String
8          get() = _currentScrambledWord
9  }
```

Trong GameFragment, cập nhật phương thức updateNextWordOnScreen() để sử dụng thuộc tính viewModel chỉ có thể đọc là currentScrambledWord.

```
 * Displays the next scrambled word on screen.
 */
private fun updateNextWordOnScreen() {
    binding.textViewUnscrambledWord.text = viewModel.currentScrambledWord
}
```

1.6. The lifecycle of a ViewModel

Trong GameViewModel.kt, hãy thêm khối init bằng câu lệnh nhật ký

```
class GameViewModel : ViewModel() {
    init {
        Log.d( tag: "GameFragment",  msg: "GameViewModel created!")
    }
```

Trong lớp GameViewModel, ghi đè phương thức onCleared(). ViewModel bị huỷ khi mảnh được liên kết bị tách rời hoặc khi hoạt động kết thúc. Ngay trước khi ViewModel bị huỷ, hệ thống sẽ thực hiện lệnh gọi lại onCleared().

Thêm câu lệnh nhật ký vào trong onCleared() để theo dõi vòng đời của GameViewModel.

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    binding = GameFragmentBinding.inflate(inflater, container,  attachToParent: false)
    Log.d( tag: "GameFragment",  msg: "GameFragment created/re-created!")
    return binding.root
}
```

Trong GameFragment, ghi đè phương thức gọi lại onDetach(). Phương thức này sẽ được gọi khi hoạt động và mảnh tương ứng bị huỷ.

```
override fun onDetach() {
    super.onDetach()
    Log.d( tag: "GameFragment",  msg: "GameFragment destroyed!")
}
```

1.7. Populate ViewModel

Tạo phương thức getNextWord() trong lớp GameViewModel

```
19      private fun getNextWord() {
20          currentWord = allWordsList.random()
21          val tempWord = currentWord.toCharArray()
22          tempWord.shuffle()
23
24          while (String(tempWord).equals(currentWord,  ignoreCase: false)) {
25              tempWord.shuffle()
26          }
27          if (wordsList.contains(currentWord)) {
28              getNextWord()
29          } else {
30              _currentScrambledWord = String(tempWord)
31              ++currentWordCount
32              wordsList.add(currentWord)
33          }
34      }
35
```

Trong lớp GameViewModel, hãy thêm một phương thức khác tên là nextWord(). Lấy từ tiếp theo trong danh sách rồi trả về true nếu số từ nhỏ hơn MAX_NO_OF_WORDS.

```
fun nextWord(): Boolean {
    return if (currentWordCount < MAX_NO_OF_WORDS) {
        getNextWord()
        true
    } else false
}
```

1.8. Dialogs

Thêm thuộc tính sao lưu vào biến score. Trong GameViewModel, hãy thay đổi nội dung khai báo biến score thành nội dung sau đây.

```
11      private var _score = 0
12      val score: Int
13          get() = _score
14
```

Thêm hàm private tên là showFinalScoreDialog()

```
private fun showFinalScoreDialog() {
    MaterialAlertDialogBuilder(requireContext())
        .setTitle(getString(R.string.congratulations))
        .setMessage(getString(R.string.you_scored, viewModel.score))
        .setCancelable(false)
        .setNegativeButton(getString(R.string.exit)) { _, _ ->
            exitGame()
        }
        .setPositiveButton(getString(R.string.play_again)) { _, _ ->
            restartGame()
        }
        .show()
}
```

1.9. Implement OnClickListener for submit button

Sửa lại onSubmitWord cho ứng dụng

```
*/
private fun onSubmitWord() {
    val playerWord = binding.textInputEditText.text.toString()

    if (viewModel.isUserWordCorrect(playerWord)) {
        setErrorTextField(false)
        if (viewModel.nextWord()) {
            updateNextWordOnScreen()
        } else {
            showFinalScoreDialog()
        }
    } else {
        setErrorTextField(true)
    }
}
```

1.10. Implement skip button

```
/*
 * Skips the current word without changing the score.
 * Increases the word count.
 */
private fun onSkipWord() {
    if (viewModel.nextWord()) {
        setErrorTextField(false)
        updateNextWordOnScreen()
    } else {
        showFinalScoreDialog()
    }
}
```

1.11. Verify the ViewModel preserves data

Để ghi nhận rằng dữ liệu ứng dụng được lưu giữ trong ViewModel khi thay đổi cấu hình ta cần ghi nhật ký vào Fragment. Để truy cập currentWordCount trong GameFragment, cần hiện phiên bản chỉ có thể đọc bằng cách sử dụng thuộc tính sao lưu.

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    Log.d( tag: "GameFragment",  msg: "Word: ${viewModel.currentScrambledWord} " +
            "Score: ${viewModel.score} WordCount: ${viewModel.currentWordCount}")
    binding = GameFragmentBinding.inflate(inflater, container,  attachToParent: false)
    Log.d( tag: "GameFragment",  msg: "GameFragment created/re-created!")
    return binding.root
}
```

1.12. Update game restart logic

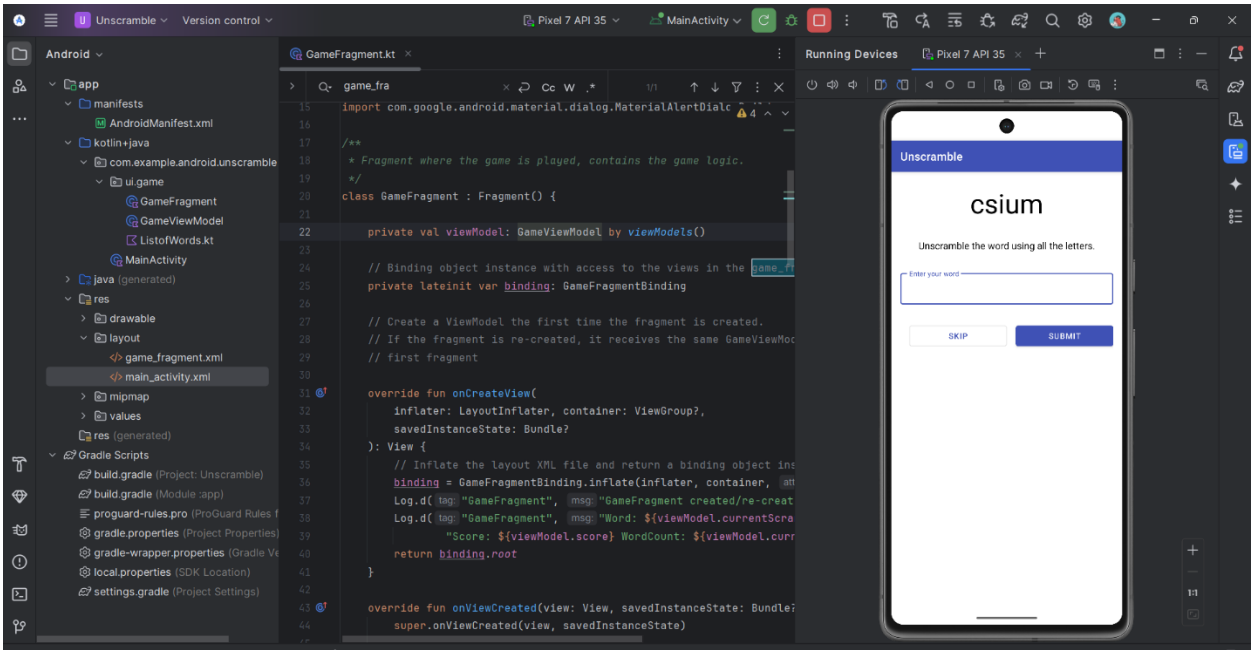
Để thiết lập lại dữ liệu ứng dụng, trong GameViewModel, hãy thêm một phương thức tên là reinitializeData(). Thiết lập điểm và số từ thành 0. Xoá danh sách từ và gọi phương thức getNextWord().

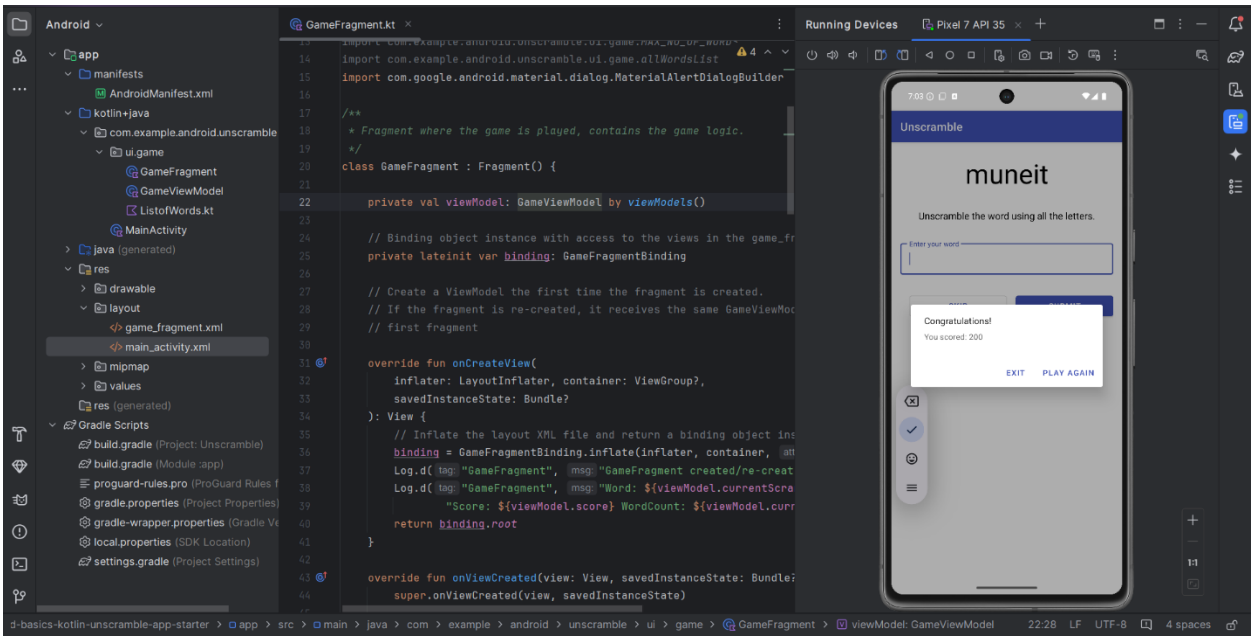
```
fun reinitializeData() {
    _score = 0
    _currentWordCount = 0
    wordsList.clear()
    getNextWord()
}
```

Gọi đến từ GameFragment

```
private fun restartGame() {
    viewModel.reinitializeData()
    setErrorTextField(false)
    updateNextWordOnScreen()
}
```

Ta được kết quả như sau :





1.13. Solution code

1.14. Summary

1.15. Learn more

2. Use LiveData with ViewModel

2.1. Before you begin

2.2. Starter app overview

Tải code về thông qua github, do đã tải sẵn từ 8.1 nên ta sửa thẳng vào code của bài lab trước

2.3. What is LiveData

LiveData là một lớp lưu giữ dữ liệu có thể quan sát được và nhận biết được vòng đời.

Một số đặc điểm của LiveData:

LiveData giữ dữ liệu; LiveData là một trình bao bọc có thể được sử dụng với bất kỳ loại dữ liệu nào.

LiveData có thể quan sát được, tức là đối tượng tiếp nhận dữ liệu sẽ nhận được thông báo khi dữ liệu được đối tượng LiveData giữ có thay đổi.

LiveData nhận biết được vòng đời. Khi bạn đính kèm một đối tượng tiếp nhận dữ liệu vào LiveData, đối tượng tiếp nhận dữ liệu này sẽ được liên kết với một LifecycleOwner (thường là một hoạt động hoặc mảnh). LiveData chỉ cập nhật những đối tượng tiếp nhận dữ liệu ở trạng thái vòng đời hoạt động, chẳng hạn như STARTED hoặc RESUMED.

2.4. Add LiveData to the curent scramble word

MutableLiveData là phiên bản có thể biến đổi của LiveData, tức là giá trị của dữ liệu được lưu trữ trong đó có thể thay đổi.

- Trong GameViewModel, thay đổi loại biến _currentScrambledWord thành MutableLiveData<String>. LiveData và MutableLiveData là các lớp chung, do vậy, bạn cần chỉ định loại dữ liệu mà các lớp đó giữ.
- Thay đổi loại biến _currentScrambledWord thành val vì giá trị của đối tượng LiveData/MutableLiveData sẽ giữ nguyên và chỉ có dữ liệu được lưu trữ trong đối tượng mới thay đổi.

```
private val _currentScrambledWord = MutableLiveData<String>()
val currentScrambledWord: LiveData<String>
    get() = _currentScrambledWord
```

Để truy cập vào dữ liệu trong đối tượng LiveData, sử dụng thuộc tính value. Trong GameViewModel bên trong phương thức getNextWord(), trong khối else, thay đổi tham chiếu của _currentScrambledWord thành _currentScrambledWord.value.

```
    // If we retry 10 times or have already used the word, pick a new word
    if (wordsList.contains(currentWord) || retryCount == 10) {
        getNextWord()
    } else {
        _currentScrambledWord.value = String(tempWord)
        ++_currentWordCount
        wordsList.add(currentWord)
    }
}
```

2.5. Attach observer to the LiveData Object

Trong GameFragment, xoá phương thức updateNextWordOnScreen() và tất cả các lệnh gọi đến phương thức đó. Bạn không cần đến phương pháp này, do bạn sẽ đính kèm đối tượng tiếp nhận dữ liệu vào LiveData.

Trong onSubmitWord(), sửa đổi khối if-else trống như sau. Phương thức hoàn chỉnh sẽ có dạng như sau.

```
private fun onSubmitWord() {
    val playerWord = binding.textInputEditText.text.toString()

    if (viewModel.isUserWordCorrect(playerWord)) {
        setErrorTextField(false)
        if (!viewModel.nextWord()) {
            showFinalScoreDialog()
        }
    } else {
        setErrorTextField(true)
    }
}
```

Đính kèm một đối tượng tiếp nhận dữ liệu cho currentScrambledWord LiveData. Trong GameFragment khi kết thúc lệnh gọi lại onViewCreated(), gọi phương thức observe() trên currentScrambledWord.

```
        R.string.word_count,  ...formatArgs: 0, MAX_NO_OF_WORDS)
viewModel.currentScrambledWord.observe()
```

Chuyển viewLifecycleOwner dưới dạng tham số đầu tiên vào phương thức observe(). viewLifecycleOwner thể hiện vòng đời của Chế độ xem mảnh. Tham số này giúp LiveData nhận biết được vòng đời của GameFragment và chỉ thông báo cho đối tượng tiếp nhận dữ liệu khi GameFragment ở trong trạng thái hoạt động (STARTED hoặc RESUMED).

Thêm một lambda làm tham số thứ hai với tham số hàm là newWord. newWord sẽ chứa giá trị của từ bị xáo trộn mới.

```
    viewModel.currentScrambledWord.observe(viewLifecycleOwner,
        { newWord -> })
}
```

Trong nội dung hàm của biểu thức lambda, gán newWord cho chế độ xem văn bản từ bị xáo trộn.

```
    viewModel.currentScrambledWord.observe(viewLifecycleOwner
    ) { newWord ->
        binding.textViewUnscrambledWord.text = newWord
    }
}
```

2.6. Attach Observer to score and word count

Bước 1: Tổng hợp điểm và số từ bằng LiveData

Trong GameViewModel, thay đổi loại của _score và biến lớp _currentWordCount thành val.

Thay đổi loại dữ liệu của các biến _score và _currentWordCount thành MutableLiveData, rồi khởi chạy các biến đó cho 0.

Thay đổi loại trường sao lưu thành LiveData<Int>

```
class GameViewModel : ViewModel(){
    private val _score = MutableLiveData( value: 0)
    val score: LiveData<Int>
        get() = _score

    private val _currentWordCount = MutableLiveData( value: 0)
    val currentWordCount: LiveData<Int>
        get() = _currentWordCount
```

Trong GameViewModel ở đầu phương thức reinitializeData(), thay đổi tham chiếu của _score và _currentWordCount thành _score.value và _currentWordCount.value tương ứng.

```
    */
    fun reinitializeData() {
        _score.value = 0
        _currentWordCount.value = 0
        wordsList.clear()
        getNextWord()
    }
```

Trong GameViewModel, bên trong phương thức nextWord(), thay đổi tham chiếu của _currentWordCount thành _currentWordCount.value!!.

```
    fun nextWord(): Boolean {
        return if (_currentWordCount.value!! < MAX_NO_OF_WORDS) {
            getNextWord()
            true
        } else false
    }
}
```

Trong GameViewModel, bên trong phương thức increaseScore() và getNextWord(), thay đổi tham chiếu của _score và _currentWordCount thành _score.value và _currentWordCount.value tương ứng. Android Studio sẽ hiển thị lỗi để bạn xem do _score không còn là số nguyên nữa mà là LiveData, bạn sẽ khắc phục lỗi này trong các bước tiếp theo.

Sử dụng hàm Kotlin plus() để tăng giá trị _score. Giá trị này bổ sung giá trị rỗng an toàn.

```
    */
    private fun increaseScore() {
        _score.value = (_score.value)?.plus(SCORE_INCREASE)
    }

    /*
```

Tương tự, sử dụng hàm Kotlin inc() để giá trị tăng thêm một bằng giá trị rỗng an toàn.

```
    private fun getNextWord() {
            return
        }

        currentWord = allWordsList.random()  // Pick a random word
        val tempWord = currentWord.toCharArray()
        tempWord.shuffle()  // Shuffle the letters

        // Retry shuffling if the word equals the original
        var retryCount = 0
        while (String(tempWord).equals(currentWord, ignoreCase: false) && retryCount < 10) {
            tempWord.shuffle()
            retryCount++
        }

        // If we retry 10 times or have already used the word, pick a new word
        if (wordsList.contains(currentWord) || retryCount == 10) {
            getNextWord()
        } else {
            _currentScrambledWord.value = String(tempWord)
            _currentWordCount.value = (_currentWordCount.value)?.inc()
            wordsList.add(currentWord)
        }
    }
```

Trong GameFragment, truy cập giá trị của score bằng thuộc tính value. Bên trong phương thức showFinalScoreDialog(), thay đổi viewModel.score thành viewModel.score.value.

```
private fun showFinalScoreDialog() {
    MaterialAlertDialogBuilder(requireContext())
        .setTitle(getString(R.string.congratulations))
        .setMessage(getString(R.string.you_scored, viewModel.score.value))
        .setCancelable(false)
        .setNegativeButton(getString(R.string.exit)) { _, _ ->
            exitGame()
        }
        .setPositiveButton(getString(R.string.play_again)) { _, _ ->
            restartGame()
        }
        .show()
}
```

Bước 2: Đính kèm đối tượng tiếp nhận dữ liệu vào điểm số và số từ

Trong GameFragment bên trong phương thức onViewCreated(), xoá mã cập nhật điểm số và chế độ xem văn bản từ.

Xoá:

```
binding.score.text = getString(R.string.score,  ...formatArgs: 0)
binding.wordCount.text = getString(
    R.string.word_count,  ...formatArgs: 0, MAX_NO_OF_WORDS)
```

Trong GameFragment ở cuối phương thức onViewCreated(), đính kèm đối tượng tiếp nhận dữ liệu cho score. Chuyển viewLifecycleOwner làm tham số đầu tiên vào đối tượng tiếp nhận dữ liệu và biểu thức lambda làm tham số thứ hai. Bên trong biểu thức lambda, chuyển điểm mới làm tham số và bên trong nội dung hàm, đặt điểm mới thành chế độ xem văn bản.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    // Setup a click listener for the Submit and Skip buttons.
    binding.submit.setOnClickListener { onSubmitWord() }
    binding.skip.setOnClickListener { onSkipWord() }
    // Update the UI
    updateNextWordOnScreen()
    // Observe the scrambledCharArray LiveData, passing in the LifecycleOwner and the obse
    viewModel.currentScrambledWord.observe(viewLifecycleOwner
    ) { newWord ->
        binding.textViewUnscrambledWord.text = newWord
    }
    viewModel.score.observe(viewLifecycleOwner
    ) { newScore ->
        binding.score.text = getString(R.string.score, newScore)
    }
}
```

Ở cuối phương thức onViewCreated(), đính kèm đối tượng tiếp nhận dữ liệu cho currentWordCount LiveData. Chuyển viewLifecycleOwner làm tham số đầu tiên vào đối tượng tiếp nhận dữ liệu và biểu thức lambda làm tham số thứ hai. Bên trong biểu thức lambda, hãy chuyển số từ mới làm tham số và trong nội dung hàm, đặt số từ mới cùng với MAX_NO_OF_WORDS thành chế độ xem văn bản.

```
        viewModel.currentWordCount.observe(viewLifecycleOwner
        ) { newWordCount ->
            binding.wordCount.text =
                getString(R.string.word_count, newWordCount, MAX_NO_OF_WORDS)
        }
    }
```

Chạy ứng dụng của bạn để xem điều kỳ diệu xảy ra. Chơi trò chơi với một vài từ. Điểm số và số từ cũng được cập nhật chính xác trên màn hình. Lưu ý rằng bạn đang không cập nhật các chế độ xem văn bản này dựa trên một số điều kiện trong mã. score và currentWordCount là LiveData và các đối tượng tiếp nhận dữ liệu tương ứng tự động được gọi khi giá trị cơ bản thay đổi.

2.7. Use LiveData with data binding

Bước 1: Thay đổi liên kết chế độ xem thành liên kết dữ liệu

Trong tệp build.gradle(Module), kích hoạt thuộc tính dataBinding trong mục buildFeatures.

Thay thế

```
buildFeatures {
    dataBinding true
}
```

Để sử dụng tính năng liên kết dữ liệu trong bất kỳ dự án Kotlin nào, bạn nên áp dụng trình bổ trợ kotlin-kapt. Bước này đã được thực hiện trong tệp build.gradle(Module).

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-kapt'
}
```

Bước 2: Chuyển đổi tệp bố cục thành bố cục liên kết dữ liệu

Mở game_fragment.xml, chọn thẻ mã.

Để chuyển đổi bố cục thành bố cục Data Binding (Liên kết dữ liệu), hãy gói thành phần gốc trong thẻ <layout>. Bạn cũng sẽ phải di chuyển các định nghĩa của không gian tên (các thuộc tính bắt đầu bằng xmlns:) sang phần tử gốc mới. Thêm thẻ <data></data> bên trong thẻ <layout> ở trên thành phần gốc. Android Studio tạo điều kiện thuận lợi để thực hiện

việc này một cách tự động: Nhấp chuột phải vào thành phần gốc (ScrollView), chọn Show Context Actions > Convert to data binding layout (Hiển thị hành động theo ngữ cảnh > Chuyển đổi sang bố cục liên kết dữ liệu)

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data>

    </data>
```

Trong GameFragment, ở đầu phương thức onCreateView(), thay đổi cách tạo bản sao của biến binding để sử dụng tính năng liên kết dữ liệu.

Thay thế

```
// Inflate the layout XML file and return a binding object instance
binding = DataBindingUtil.inflate(inflater, R.layout.game_fragment, container,
```

2.8. Add data binding variables

Trong game_fragment.xml, bên trong thẻ <data>, thêm một thẻ con có tên là <variable>, khai báo một thuộc tính có tên là gameViewModel và thuộc loại GameViewModel. Bạn sẽ dùng thẻ này để liên kết dữ liệu trong ViewModel với bố cục.

Bên dưới khai báo gameViewModel, thêm một biến khác vào bên trong thẻ <data> của loại Integer và đặt tên cho biến đó là maxNoOfWords. Bạn sẽ sử dụng biến này để liên kết với biến trong ViewModel để lưu trữ số từ trên mỗi lượt chơi.

```
    <data>
        <variable
            name="gameViewModel"
            type="com.example.android.unscramble.ui.game.GameViewModel" />
    </data>
```

Trong GameFragment ở đầu phương thức onViewCreated(), hãy khởi chạy các biến bố cục gameViewModel và maxNoOfWords.

```
binding.gameViewModel = viewModel

binding.maxNoOfWords = MAX_NO_OF_WORDS
```

LiveData có thể quan sát nhận biết được vòng đời nên bạn phải chuyển chủ sở hữu vòng đời vào bố cục. TrongGameFragment , bên trongonViewCreated() ở dưới phần khởi chạy của các biến liên kết, thêm vào mã sau.

```
binding.lifecycleOwner = viewLifecycleOwner
```

2.9. Use binding expressions

Bước 1: Thêm biểu thức liên kết vào từ hiện tại

Trong game_fragment.xml, thêm thuộc tính text vào chế độ xem văn bản textView_unscrambled_word. Sử dụng biến bố cục mới là gameViewModel và gán @{gameViewModel.currentScrambledWord} cho thuộc tính text.

```
<TextView
    android:id="@+id/textView_unscrambled_word"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="@dimen/default_margin"
    android:layout_marginBottom="@dimen/default_margin"
    android:textAppearance="@style/TextAppearance.MaterialComponents.H
    android:text="@{gameViewModel.currentScrambledWord}"
```

Trong GameFragment, xoá mã đối tượng tiếp nhận dữ liệu LiveData cho currentScrambledWord: Bạn không còn cần đến mã đối tượng tiếp nhận dữ liệu trong mảnh nữa. Bố cục sẽ trực tiếp nhận được thông tin cập nhật về những thay đổi đối với LiveData.

Bước 2: Thêm biểu thức liên kết vào điểm số và số từ

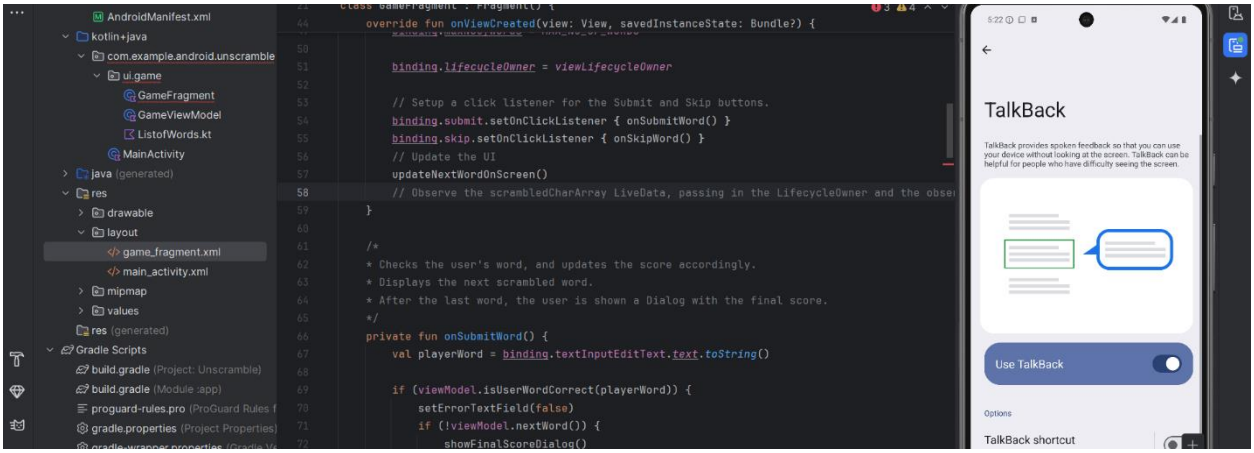
Trong bước này, bạn sẽ thêm biểu thức liên kết vào chế độ xem điểm số và văn bản số từ, chuyển tham số tài nguyên. Bước này tương tự như bước bạn đã làm cho textView_unscrambled_word ở trên.

Trong game_fragment.xml, cập nhật thuộc tính text cho chế độ xem văn bản word_count bằng biểu thức liên kết sau. Sử dụng tài nguyên chuỗi word_count và chuyển vào gameViewModel.currentWordCount còn maxNoOfWords làm tham số tài nguyên.

Xoá đối tượng tiếp nhận dữ liệu LiveData khỏi GameFragment. Bạn không cần các đối tượng tiếp nhận dữ liệu đó nữa, biểu thức liên kết sẽ cập nhật giao diện người dùng khi LiveData tương ứng thay đổi.

2.10. Test Unscramble app with Talkback enabled

Bật TalkBack




```
private val _currentScrambledWord = MutableLiveData<String>()
val currentScrambledWord: LiveData<Spannable> = Transformations.map(_currentScrambledWord)
    if (it == null) {
        SpannableString( source: "")
    } else {
        val scrambledWord = it.toString()
        val spannable: Spannable = SpannableString(scrambledWord)
        spannable.setSpan(
            TtsSpan.VerbatimBuilder(scrambledWord).build(),
            0,
            scrambledWord.length,
            Spannable.SPAN_INCLUSIVE_INCLUSIVE
        )
        spannable
    }
}
```

Chạy app ta được kết quả cuối cùng

2.11. Delete unused code

2.12. Solution Code

2.13. Summary

2.14. Learn more