برنامه نویسی دستگاه های سیار (CE364)

جلسه یازدهم:
مدیریت حالت برنامه

سجاد شیرعلی شهرضا

پاییز 1401

دوشنبه، 21 آذر 1401

اطلاع رسانی

- بخشهای مرتبط با این جلسه:

- Unit 3: Navigation:
  - Pathway 3: Architecture components

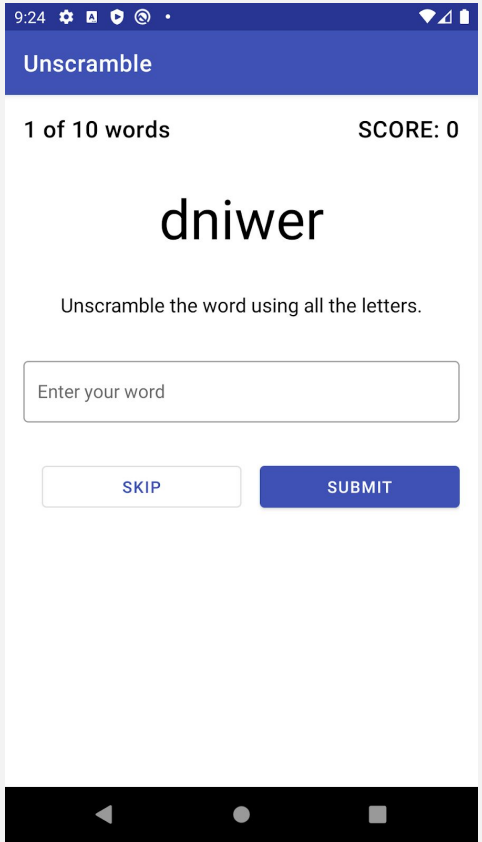
سوال؟

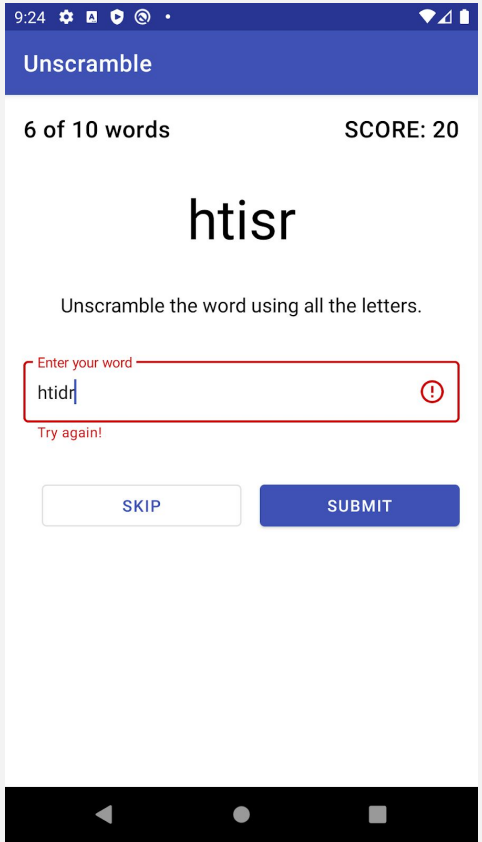
مدل نما

مقدمه

- جت پک: مجموعه ای از کتابخانه ها برای راحت تر کردن توسعه نرم افزار برای اندروید
- مولفه های معماری برنامه (Android Architecture Components)
  - بخشی از جت پک
  - مجموعه ای از قواعد طراحی

برنامه نهایی این جلسه: درست کردن کلمه درهم ریخته





محدودیت های نسخه اولیه

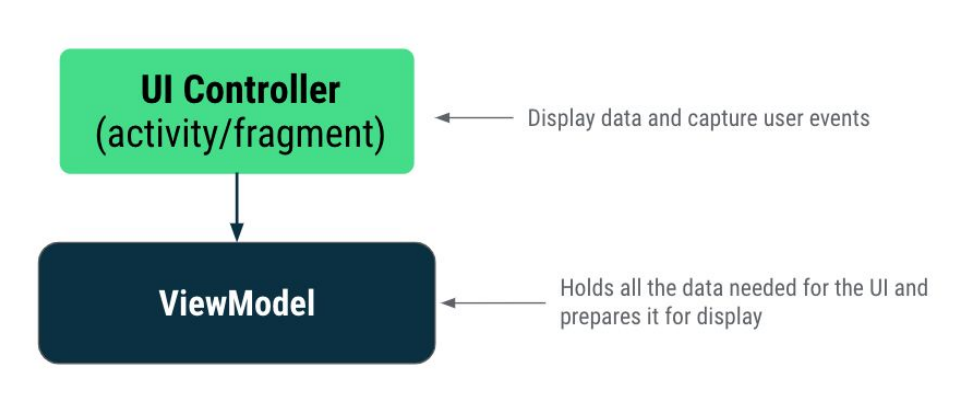
- بررسی نکردن جواب کاربر
- تمام نشدن بازی پس از 10 کلمه
- ذخیره نکردن حالت برنامه
  - شروع دوباره برنامه در صورت چرخش گوشی

معماری برنامه

- هدف معماری: تخصیص وظایف میان کلاس ها
- مزیت معماری خوب: توسعه راحت برنامه برای افزودن ویژگی های جدید
- دو اصل معمول معماری:
  - تفکیک وظایف: تقسیم برنامه به کلاس های مختلف با وظایف مجزای
  - اجرای رابط کاربری از روی یک مدل

اجرای رابط کاربری از روی یک مدل

- مدل: مولفه نگه دارنده اطلاعات برنامه
- مستقل از نماها و چرخه حیات آنها
- کلاس های اصلی در معماری اندروید:
  - کنترل کننده رابط کاربری، ViewModel, LiveData, Room



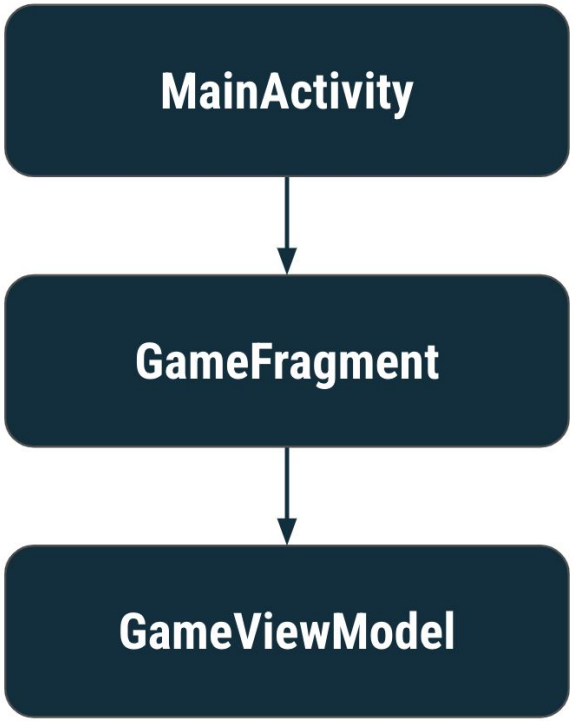
تفاوت مدل نما و کنترل کننده نما

- کنترلر نما:
  - نمایش بخش های مختلف نما برای کاربر
  - پردازش اتفاقات مرتبط با آن (مثلا فشرده شدن یک کلید)
  - توسط سیستم بر حسب شرایط ایجاد و یا از بین برده می شود
  - نباید حاوی داده و وضعیت
- مدل نما:
  - نگه داری اطلاعات مربوط به برنامه و کاربر
  - نما از روی این اطلاعات ایجاد خواهد شد
  - اطلاعات به خاطر ایجاد یا از بین رفتن نما، تغییر نمی کنند.

سوال؟

استفاده از مدل نما

اضافه کردن مدل نما



- اضافه کردن پیش نیاز مدل نما در build.gradle

```
// ViewModel
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0'
```

- اضافه کردن کلاس GameViewModel

```
class GameViewModel : ViewModel() {
}
```

- ایجاد یک نمونه از آن در قطعه

```
private val viewModel: GameViewModel by viewModels()
```

محول کردن دارایی (Property Delegation)

- تعریف متغیر قابل تغییر (var): هر دو تابع set و get تعریف می شود
- تعریف متغیر غیر قابل تغییر (val): تنها تابع get تعریف می شود
- امکان محول کردن تعریف دارایی به یک کلاس نماینده (delegate class)

```
// Syntax for property delegation
var <property-name> : <property-type> by <delegate-class>()
```

انتقال داده ها به کلاس مدل نما

```
class GameViewModel : ViewModel() {

    private var score = 0
    private var currentWordCount = 0
    private var currentScrambledWord = "test"
...
```

دارایی پشتوانه (Backing Property)

- تعریف تابع برای بازگرداندن مقدار دیگر غیر از مقدار اصلی

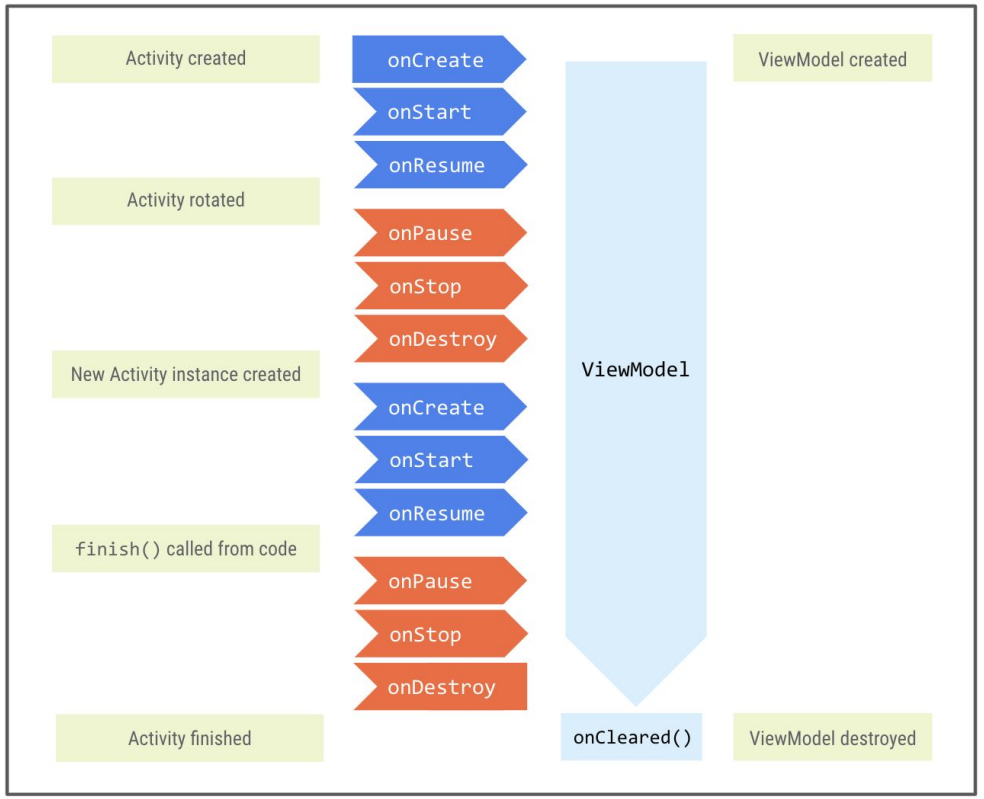
```
// Declare private mutable variable that can only be modified
// within the class it is declared.
private var _count = 0

// Declare another public immutable field and override its getter method.
// Return the private property's value in the getter method.
// When count is accessed, the get() function is called and
// the value of _count is returned.
val count: Int
    get() = _count
```

انتقال داده لغت فعلی به مدل نما

```
private var _currentScrambledWord = "test"
val currentScrambledWord: String
    get() = _currentScrambledWord
```

```
private fun updateNextWordOnScreen() {
    binding.textViewUnscrambledWord.text = viewModel.currentScrambledWord
}
```

چرخه زندگی مدل نما



اجرای کد در هنگام ایجاد یک شی

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

    ...
}
```

اتمام یک نمونه مدل نما

- مدل نما از بین می رود وقتی:
  - قطعه مرتبط به آن از آن قطع می شود
  - فعالیت اتمام می شود
- قبل از بین رفتن یک نمونه: تابع onCleared اجرا می شود

```
override fun onCleared() {
    super.onCleared()
    Log.d("GameFragment", "GameViewModel destroyed!")
}
```

- برای قطعه:
  - تابع onDetach قبل از بین رفتن قطعه و فعالیت

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

پیاده سازی لغت انتخابی

```
private var wordsList: MutableList<String> = mutableListOf()
private lateinit var currentWord: String
```

```
/*
* Updates currentWord and currentScrambledWord with the next word.
*/
private fun getNextWord() {
   currentWord = allWordsList.random()
   val tempWord = currentWord.toCharArray()
   tempWord.shuffle()

   while (String(tempWord).equals(currentWord, false)) {
       tempWord.shuffle()
   }
   if (wordsList.contains(currentWord)) {
       getNextWord()
   } else {
       _currentScrambledWord = String(tempWord)
       ++_currentWordCount
       wordsList.add(currentWord)
   }
}
```

تابع بررسی اتمام بازی

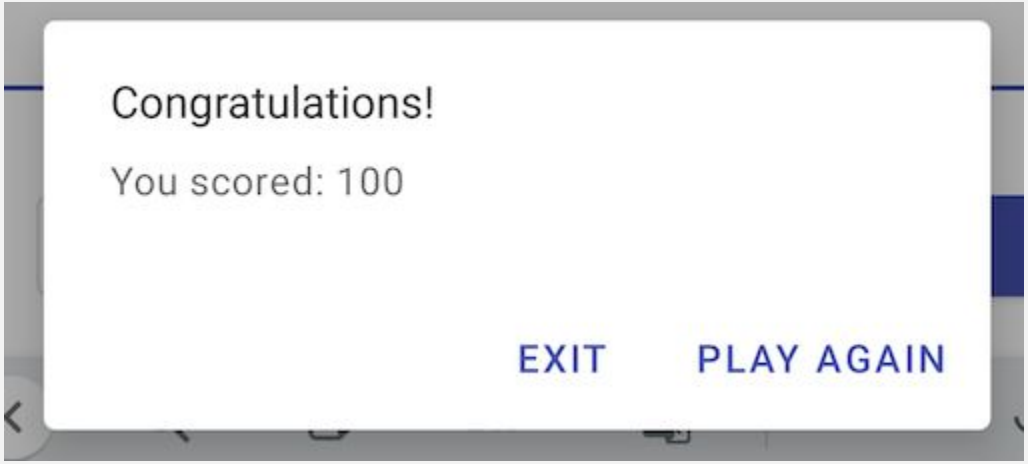
```
/*
* Returns true if the current word count is less than MAX_NO_OF_WORDS.
* Updates the next word.
*/
fun nextWord(): Boolean {
    return if (currentWordCount < MAX_NO_OF_WORDS) {
        getNextWord()
        true
    } else false
}
```

سوال؟

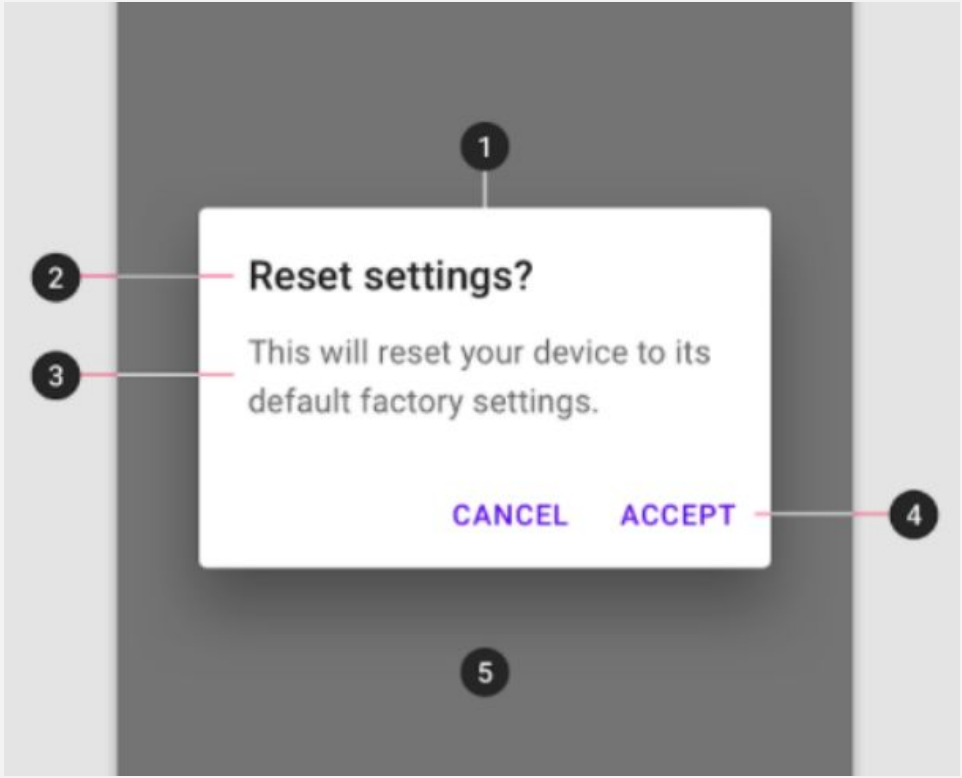
نمایش یک دیالوگ

هدف

- اعلام اتمام بازی



ساختار یک دیالوگ



1) کل دیالوگ

2) عنوان (اختیاری)

3) پیغام

4) دکمه های متنی

نمایش امتیاز نهایی

```
private var _score = 0
val score: Int
    get() = _score
```

- تعریف امتیاز در مدل نما

```
/*
* Creates and shows an AlertDialog with the final score.
*/
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()
}
```

- نمایش دیالوگ در قطعه

پردازش فشرده شدن کلید ارسال

```
private fun onSubmitWord() {
    if (viewModel.nextWord()) {
        updateNextWordOnScreen()
    } else {
        showFinalScoreDialog()
    }
}
```

بررسی جواب کاربر

```
private fun increaseScore() {
    _score += SCORE_INCREASE
}
```

```
fun isUserWordCorrect(playerWord: String): Boolean {
    if (playerWord.equals(currentWord, true)) {
        increaseScore()
        return true
    }
    return false
}
```

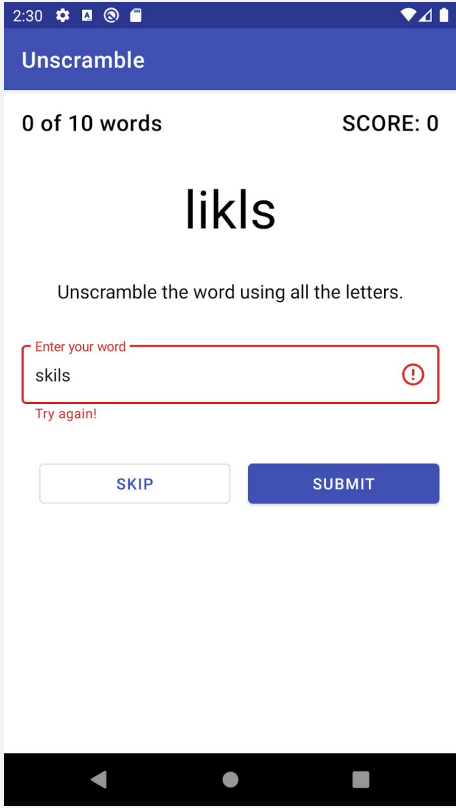
نمایش خطا در یک متن ورودی


```
// Set error text
passwordLayout.error = getString(R.string.error)

// Clear error text
passwordLayout.error = null
```

```
private fun setErrorTextField(error: Boolean) {
    if (error) {
        binding.textField.isErrorEnabled = true
        binding.textField.error = getString(R.string.try_again)
    } else {
        binding.textField.isErrorEnabled = false
        binding.textInputEditText.text = null
    }
}
```

بررسی درست بودن جواب هنگام ارسال

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

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



امکان گذشتن از یک کلمه

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

امکان بازی مجدد

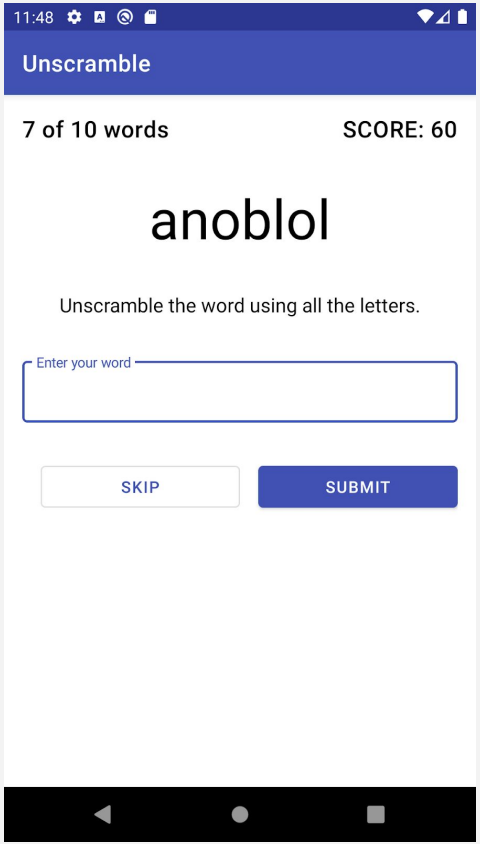
```
/*
* Re-initializes the game data to restart the game.
*/
fun reinitializeData() {
    _score = 0
    _currentWordCount = 0
    wordsList.clear()
    getNextWord()
}
```

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

سوال؟

داده زنده

هدف



- به روز رسانی امتیاز کل و شماره کلمه در طول بازی

داده زنده

- یک نگه دارنده مقدار که متناسب با چرخه زندگی برنامه، تغییر می کند
- یک پوشش داده است که می تواند هر نوع داده ای را نگه دارد
- قابل مشاهده است: مشاهده کننده از تغییر آن باخبر می شود
- از چرخه زندگی آگاه است: فقط مشاهده کننده ای را خبر می کند که فعال است

تبدیل لغت فعلی به داده زنده در مدل نما

```
private val _currentScrambledWord = MutableLiveData<String>()
```

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

```
private fun getNextWord() {
 ...
    } else {
        _currentScrambledWord.value = String(tempWord)
        ...
    }
}
```

اضافه کردن یک مشاهده کَر

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

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

```
// Observe the scrambledCharArray LiveData, passing in the LifecycleOwner and the obse
viewModel.currentScrambledWord.observe(viewLifecycleOwner,
    { newWord ->
        binding.textViewUnscrambledWord.text = newWord
    })
```

تبدیل امتیاز و تعداد کلمات به داده زنده

```
private val _score = MutableLiveData(0)
val score: LiveData<Int>
   get() = _score

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

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

```
fun nextWord(): Boolean {
    return if (_currentWordCount.value!! < MAX_NO_OF_WORDS) {
           getNextWord()
           true
       } else false
   }
private fun increaseScore() {
    _score.value = (_score.value)?.plus(SCORE_INCREASE)
}
```

تبدیل امتیاز و تعداد کلمات به داده زنده (ادامه)

```
private fun getNextWord() {
    ...
     } else {
         _currentScrambledWord.value = String(tempWord)
         _currentWordCount.value = (_currentWordCount.value)?.inc()
         wordsList.add(currentWord)
        }
    }
```

```
private fun showFinalScoreDialog() {
    MaterialAlertDialogBuilder(requireContext())
         .setTitle(getString(R.string.congratulations))
         .setMessage(getString(R.string.you_scored, viewModel.score.value))
         ...
         .show()
}
```

اضافه کردن مشاهده گَرها

- حذف نمایش و ارتباط مستقیم داده به نما

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

- اضافه کردن مشاهده گر

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

سوال؟

اتصال داده

اتصال داده (Data Binding)

- اتصال یک داده به یک عنصر چیدمان
  - اتصال داده به نما
  - اتصال نما به داده
- جزیی از اندروید جت پک

```
android:text="@{gameViewModel.currentScrambledWord}"
```

- عدم نیاز به درج بسیاری از توابع برای به روزرسانی رابط کاربری در برنامه
  - ساده تر شده برنامه

فعال کردن اتصال داده

- فعال کردن در build.gradle

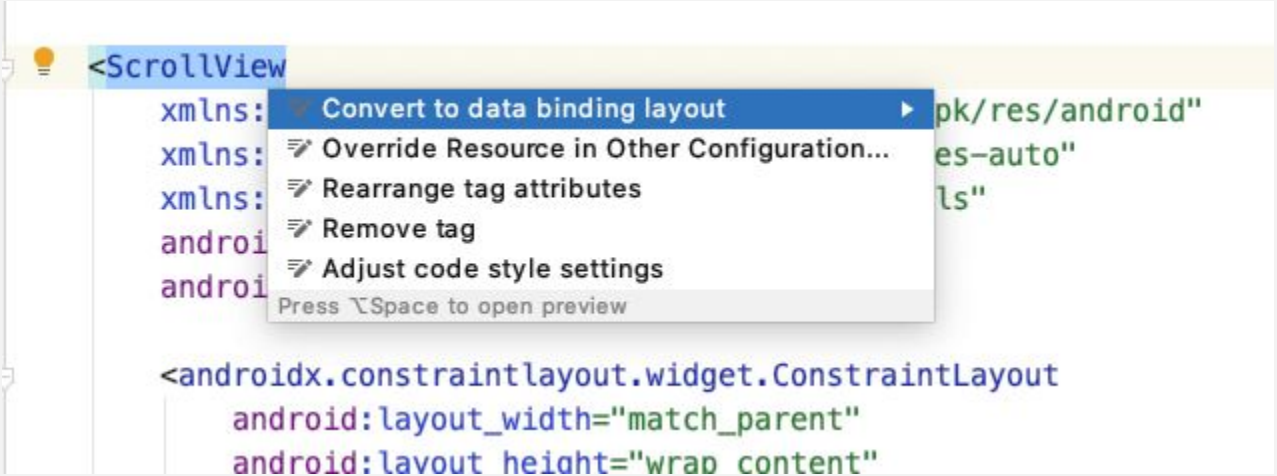
```
buildFeatures {
    dataBinding = true
}
```

- استفاده از پلاگین مربوطه

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

تبدیل خودکار نما برای پشتیبانی از اتصال داده

- انتخاب Show Context Actions > Convert to data binding layout پس از کلیک راست بر روی عنصر ریشه (ScrollView)



نمونه نمای با پشتیبانی از اتصال داده

```
<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>

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <androidx.constraintlayout.widget.ConstraintLayout
          ...
        </androidx.constraintlayout.widget.ConstraintLayout>
    </ScrollView>
</layout>
```

تبدیل نوع ایجاد نما

- جایگزینی

```
binding = GameFragmentBinding.inflate(inflater, container, false)
```

- با

```
binding = DataBindingUtil.inflate(inflater, R.layout.game_fragment, container, false)
```

تعریف اتصال داده

- تعریف دارایی برای مدل نما در game_fragment.xml

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

- تعریف متغیر

```
<data>
    ...
    <variable
        name="maxNoOfWords"
        type="int" />
</data>
```

مقداردهی به متغیرهای اتصال داده

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

    binding.gameViewModel = viewModel

    binding.maxNoOfWords = MAX_NO_OF_WORDS
...
}
```

```
// Specify the fragment view as the lifecycle owner of the binding.
// This is used so that the binding can observe LiveData updates
binding.lifecycleOwner = viewLifecycleOwner
```

اتصال داده در چیدمان

```
<TextView android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@{user.firstName}" />
```

```
<TextView
   android:id="@+id/textView_unscrambled_word"
   ...
   android:text="@{gameViewModel.currentScrambledWord}"
   .../>
```

- برای لغت فعلی

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

- حذف مشاهده گر

استفاده از دارایی های برنامه در اتصال داده

```
android:padding="@{@dimen/largePadding}"
```

```
android:text="@{@string/example_resource(user.lastName)}"
```

```
<string name="example_resource">Last Name: %s</string>
```

اتصال داده برای امتیاز و شماره کلمه

```
<TextView
   android:id="@+id/word_count"
   ...
   android:text="@{@string/word_count(gameViewModel.currentWordCount, maxNoOfWords)}"
   .../>
```

```
<TextView
    android:id="@+id/score"
    ...
    android:text="@{@string/score(gameViewModel.score)}"
    ... />
```

خواندن کلمه درهم ریخته به صورت حرف به حرف

- کاربران با مشکل بینایی ممکن است از Talkback استفاده کنند

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

سوال؟