1. LiveData c.d.

2. Czego się nauczysz

- Jak korzystać z elementów Data Binding Library.
- Jak zintegrować `ViewModel` z data binding.
- ak zintegrować `LiveData` z data binding.
- Jak używać listener bindings w celu zastąpienia detektorów kliknięć we fragmencie.
- Jak dodać formatowanie ciągów do wyrażeń wiążących dane. (string formatting, data-binding expressions).

Co będziesz robić

- Widoki w układach GuessTheWord komunikują się pośrednio z obiektami ViewModel, wykorzystując kontrolery interfejsu użytkownika (fragmenty) do przekazywania informacji. W tym ćwiczeniu łączymy widoki aplikacji z obiektami ViewModel, dzięki czemu widoki komunikują się bezpośrednio z obiektami ViewModel.
- Zmodyfikujesz aplikację, tak aby używała LiveData jako źródła wiązania danych. Po tej zmianie obiekty LiveData powiadamiają interfejs użytkownika o zmianach danych, a metody obserwatora LiveData nie będą już potrzebne.

3. Wprowadzenie

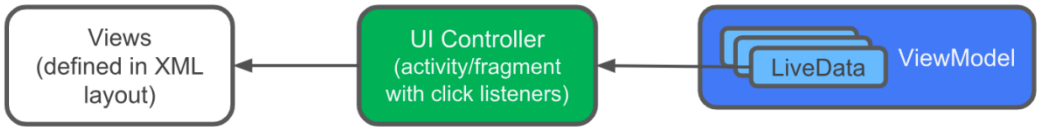
W tym ćwiczeniu udoskonalimy aplikację GuessTheWord poprzez zintegrowanie powiązania danych z LiveData w obiektach ViewModel. To automatyzuje komunikację między widokami w układzie a obiektami ViewModel i pozwala uprościć kod przy użyciu `LiveData`.

4. Zadanie: Dodaj powiązanie danych ViewModel

W poprzednim ćwiczeniu użyto powiązania danych jako bezpiecznego typu dostępu do widoków w aplikacji GuessTheWord. Ale prawdziwą siłą wiązania danych jest robienie tego, co sugeruje nazwa: wiązanie danych *bezpośrednio* do obiektów widoku w aplikacji..

Obecna architektura aplikacji

W Twojej aplikacji widoki są zdefiniowane w układzie XML, a dane dla tych widoków są przechowywane w obiektach `ViewModel` Pomiędzy każdym widokiem a odpowiadającym mu `ViewModel` znajduje się kontroler interfejsu użytkownika, który działa jak przekaźnik między nimi.



Na przykład:

- Przycisk Got It jest zdefiniowany jako widok przycisku w pliku układu `game_fragment.xml`.
- Gdy użytkownik stuknie przycisk **Got It** button, , detektor kliknięć w fragmencie `GameFragment` wywołuje odpowiedni detektor kliknięć w `GameViewModel`.
- Wynik jest aktualizowany w `GameViewModel`.

Widok przycisku i GameViewModel nie komunikują się bezpośrednio - potrzebują nasłuchiwania kliknięć w `GameFragment`.

ViewModel przekazany do powiązania danych (data binding)

Byłoby łatwiej, gdyby widoki w układzie komunikowały się bezpośrednio z danymi w obiektach `ViewModel` , bez polegania na kontrolerach interfejsu użytkownika jako pośrednikach.

Obiekty `ViewModel` przechowują wszystkie dane interfejsu użytkownika w aplikacji GuessTheWord . Przekazując obiekty `ViewModel` do powiązania danych, można zautomatyzować część komunikacji między widokami a obiektami `ViewModel`.

W tym zadaniu kojarzysz klasy `GameViewModel` i `ScoreViewModel` z odpowiadającymi im układami XML (XML layouts). Skonfigujesz również powiązania detektora do obsługi zdarzeń kliknięcia. (listener bindings, click events)

Step 1: Dodaj powiązanie danych dla GameViewModel

W tym kroku skojarzysz `GameViewModel` z odpowiednim plikiem układu, `game_fragment.xml`.

1. W `game_fragment.xml` dodaj zmienną wiążącą dane typu `GameViewModel`. Jeśli masz błędy w Android Studio, wyczyść i odbuduj projekt.

```
<layout ...>

   <data>

       <variable
           name="gameViewModel"

type="com.example.android.guesstheword.screens.game.GameViewModel" />
   </data>

   <androidx.constraintlayout...
```

2. W pliku `GameFragment` przekaż `GameViewModel` do powiązania danych.

Aby to zrobić, przypisz `viewModel` do zmiennej `binding.gameViewModel` , którą zadeklarowałeś w poprzednim kroku. Umieść ten kod wewnątrz `onCreateView()`,po zainicjowaniu **`viewModel`** Jeśli masz błędy w Android Studio, wyczyść i odbuduj projekt.

```
// Set the viewmodel for databinding - this allows the bound layout access
// to all the data in the ViewModel
binding.gameViewModel = viewModel
```

Step 2: Użuj listener bindings obsługi zdarzeń event handling

Listener bindings są wyrażeniami wiążącymi, które są uruchamiane po wywołaniu zdarzeń takich jak `onClick()`, `onZoomIn()`, or `onZoomOut()` *Listener bindings* są zapisywane jako wyrażenia lambda.

tworzy detektor i ustawia detektor w widoku. Kiedy zdarza się zdarzenie nasłuchiwane, słuchacz ocenia wyrażenie lambda. Powiązania programu nasłuchującego działają z wtyczką Gradle dla systemu Android w wersji 2.0 lub nowszej. Aby dowiedzieć się więcej, przeczytaj Layouts and binding expressions.

W tym kroku, zamieniasz detektory kliknięć w `GameFragment` na listener bindings w pliku `game_fragment.xml`.

1. W `game_fragment.xml`, dodaj atrybut `onClick` do przycisku **`skip_button`**. Zdefiniuj wyrażenie wiążące i wywołaj metodę `onSkip()` w `GameViewModel`. To wyrażenie wiązania jest nazywane *listener binding*.

```
<Button
   android:id="@+id/skip_button"
   ...
   android:onClick="@{() -> gameViewModel.onSkip()}"
   ... />
```

2. Podobnie powiąż zdarzenie click przycisku **`correct_button`** z metodą **`onCorrect`**`()` w `GameViewModel`.

```
<Button
   android:id="@+id/correct_button"
   ...
   android:onClick="@{() -> gameViewModel.onCorrect()}"
   ... />
```

3. Powiąż zdarzenie click przycisku **`end_game_button`** z metodą **`onGameFinish`**`()` w `GameViewModel`.

```
<Button
   android:id="@+id/end_game_button"
   ...
   android:onClick="@{() -> gameViewModel.onGameFinish()}"
   ... />
```

4. W `GameFragment`, usuń instrukcje ustawiające detektory kliknięć i usuń funkcje wywoływane przez detektory kliknięć. Już ich nie potrzebujesz.

Kod do usunięcia:

```
binding.correctButton.setOnClickListener { onCorrect() }
binding.skipButton.setOnClickListener { onSkip() }
binding.endGameButton.setOnClickListener { onEndGame() }

/** Methods for buttons presses **/
private fun onSkip() {
   viewModel.onSkip()
}
private fun onCorrect() {
   viewModel.onCorrect()
}
private fun onEndGame() {
   gameFinished()
}
```

Step 3: Dodaj powiązanie danych dla ScoreViewModel

W tym kroku skojarzysz `ScoreViewModel` z odpowiednim plikiem układu, `score_fragment.xml`.

1. W pliku `score_fragment.xml` dodaj zmienną wiążącą typu `ScoreViewModel`. Ten krok jest podobny do tego, co zrobiłeś dla `GameViewModel` powyżej.

```
<layout ...>
   <data>
       <variable
           name="scoreViewModel"

type="com.example.android.guesstheword.screens.score.ScoreViewModel" />
   </data>
   <androidx.constraintlayout.widget.ConstraintLayout
```

2. W `score_fragment.xml`, dodaj atrybut `onClick` do przycisku **`play_again_button`**. Zdefiniuj powiązanie detektora i wywołaj metodę **`onPlayAgain()`** w `ScoreViewModel`.

```
<Button
   android:id="@+id/play_again_button"
   ...
   android:onClick="@{() -> scoreViewModel.onPlayAgain()}"
   ... />
```

3. W `ScoreFragment`, wewnątrz `onCreateView()`,zainicjuj **`viewModel`**. Następnie zainicjuj zmienną powiązania `binding.scoreViewModel`.

```
viewModel = ...
binding.scoreViewModel = viewModel
```

4. W `ScoreFragment`, usuń kod, który ustawia detektor kliknięć dla `playAgainButton`. Jeśli Android Studio wyświetla błąd, wyczyść i odbuduj projekt.

Kod do usunięcia:

```
binding.playAgainButton.setOnClickListener {  viewModel.onPlayAgain()  }
```

5. Uruchom aplikację. Aplikacja powinna działać jak poprzednio, ale teraz widoki przycisków komunikują się bezpośrednio z obiektami ViewModel. Widoki nie komunikują się już za pośrednictwem procedur obsługi kliknięć przycisków w `ScoreFragment`.

Rozwiązywanie problemów z komunikatami o błędach powiązania danych

Gdy aplikacja korzysta z powiązania danych, proces kompilacji generuje klasy pośrednie, które są używane do powiązania danych. Aplikacja może zawierać błędy, których nie wykrywa Android Studio, dopóki nie spróbujesz jej skompilować, więc nie widzisz ostrzeżeń ani czerwonego kodu podczas pisania kodu. Ale w czasie kompilacji pojawiają się tajemnicze błędy pochodzące z generowanych klas pośrednich.

Jeśli pojawi się tajemniczy komunikat o błędzie:

1. Przyjrzyj się uważnie komunikatowi w okienku kompilacji Android Studio. Jeśli zobaczysz lokalizację, która kończy się wiązaniem danych, wystąpił błąd w powiązaniu danych.
2. W pliku XML układu sprawdź, czy nie występują błędy w atrybutach onClick, które korzystają z powiązania danych. Poszukaj funkcji wywoływanej przez wyrażenie lambda i upewnij się, że istnieje.
3. W sekcji `<data>` pliku XML sprawdź pisownię zmiennej wiążącej dane.

Na przykład zwróć uwagę na błąd w pisowni nazwy funkcji `onCorrect()` w następującej wartości atrybutu:

```
android:onClick="@{() -> gameViewModel.onCorrectx()}"
```

Zwróć także uwagę na błąd pisowni `gameViewModel` w sekcji `<data>` pliku XML:

```
<data>
   <variable
       name="gameViewModelx"
       type="com.example.android.guesstheword.screens.game.GameViewModel"
/>
</data>
```

Android Studio nie wykrywa takich błędów, dopóki nie skompilujesz aplikacji, a następnie kompilator wyświetli komunikat o błędzie, taki jak:

```
error: cannot find symbol
import
com.example.android.guesstheword.databinding.GameFragmentBindingImpl"
```

```
symbol:   class GameFragmentBindingImpl
location: package com.example.android.guesstheword.databinding
```

5. Zadanie: dodaj LiveData do powiązania danych

Powiązanie danych działa dobrze z LiveData używanym z obiektami ViewModel. Teraz, gdy dodałeś powiązanie danych do obiektów ViewModel, jesteś gotowy do włączenia `LiveData`.

W tym zadaniu zmienisz aplikację GuessTheWord, aby używała LiveData jako źródła powiązania danych, aby powiadamiać interfejs użytkownika o zmianach danych, bez korzystania z metod obserwatora `LiveData`.

Step 1: Dodaj word LiveData do pliku game_fragment.xml

W tym kroku powiąż bieżący widok word text view bezpośrednio z obiektem `LiveData` w `ViewModel`.

1. W `game_fragment.xml`, dodaj atrybut `android:text` do widoku **`word_text`**.

Ustaw go na obiekt `LiveData` , `word` z `GameViewModel`, używając zmiennej powiązania, **`gameViewModel`**.

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

Zauważ, że nie musisz używać `word.value`. Zamiast tego możesz użyć rzeczywistego obiektu `LiveData` Obiekt `LiveData` wyświetla bieżącą wartość `word`. Jeśli wartość `word` wynosi null, object `LiveData` wyświetla pusty ciąg.

2. W `GameFragment`, w `onCreateView()`,po zainicjowaniu `gameViewModel`, ustaw bieżącą aktywność jako właściciela zmiennej powiązania w cyklu życia . To określa zakres powyższego obiektu LiveData, umożliwiając obiektowi automatyczną aktualizację widoków w układzie, `game_fragment.xml`.

```
binding.gameViewModel = ...
// Specify the current activity as the lifecycle owner of the binding.
// This is used so that the binding can observe LiveData updates
binding.lifecycleOwner = this
```

3. W `GameFragment`, usuń obserwatora dla `LiveData word`.

Kod do usunięcia:

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

4. Uruchom aplikację i zagraj w grę. Teraz bieżące słowo jest aktualizowane bez metody obserwatora w kontrolerze interfejsu użytkownika.

Step 2: Dodaj score LiveData do pliku score_fragment.xml

W tym kroku wiążesz `LiveData score` z widokiem score text w score fragment.

1. W `score_fragment.xml`, dodaj atrybut **`android:text`** do widoku tekstu wyniku. Przypisz **`scoreViewModel.score`** do atrybutu **`text`**. Ponieważ **`score`** jest liczbą całkowitą, przekonwertuj go na ciąg za pomocą **`String.valueOf()`**.

```
<TextView
   android:id="@+id/score_text"
   ...
   android:text="@{String.valueOf(scoreViewModel.score)}"
   ... />
```

2. W `ScoreFragment`, po zainicjowaniu *`scoreViewModel`*, ustaw bieżącą aktywność jako właściciela cyklu życia zmiennej powiązania.

```
binding.scoreViewModel = ...
// Specify the current activity as the lifecycle owner of the binding.
// This is used so that the binding can observe LiveData updates
binding.lifecycleOwner = this
```

3. W `ScoreFragment`, usuń obserwatora dla obiektu score.

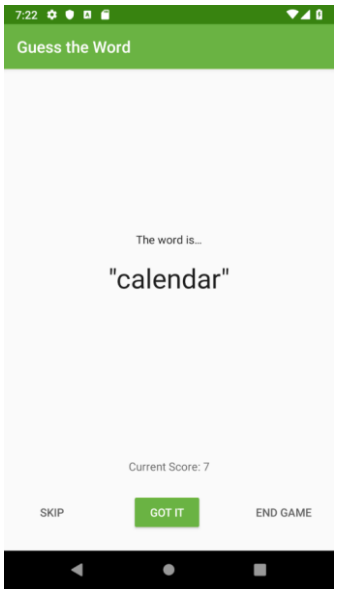
Kod do usunięcia:

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

4. Uruchom aplikację i zagraj w grę. Zauważ, że wynik we fragmencie score fragment jest wyświetlany poprawnie, bez obserwatora we fragmencie score fragment.

Step 3: Dodaj formatowanie ciągu z powiązaniem danych

W układzie można dodać formatowanie ciągów wraz z powiązaniem danych. W tym zadaniu formatujesz bieżące słowo, aby dodać wokół niego cudzysłowy. Sformatujesz również ciąg wyniku, aby poprzedzić go bieżącym wynikiem, jak pokazano na poniższym obrazku.



1. W `string.xml`, dodaj następujące ciągi, których użyjesz do sformatowania widoków `word` i `score`. `%s` i `%d` są symbolami zastępczymi dla bieżącego word i score.

```
<string name="quote_format">\"%s\"</string>
<string name="score_format">Current Score: %d</string>
```

2. W `game_fragment.xml`, zaktualizuj atrybut`text` widoku tekstowego **`word_text`** , aby korzystać z zasobu **`quote_format`**. Przekaż w `gameViewModel.word`. To przekazuje bieżące słowo jako argument do ciągu formatującego.

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

3. Sformatuj widok `score` text view podobny do tekstu **`word_text`**. W `game_fragment.xml`, dodaj atrybut `text` do widoku **`score_text`** Użyj zasobu **`score_format`**, który pobiera jeden argument liczbowy reprezentowany przez symbol zastępczy `%d`. Przekaż obiekt `LiveData` , `score`, jako argument do tego ciągu formatującego.

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

4. W klasie `GameFragment` w metodzie `onCreateView()` usuń kod obserwatora `score`.

Kod do usunięcia:

```
viewModel.score.observe(this, Observer { newScore ->
    binding.scoreText.text = newScore.toString()
})
```

5. Wyczyść, odbuduj i uruchom aplikację, a następnie zagraj w grę. Zauważ, że bieżące słowo i wynik są sformatowane na ekranie gry.