# 1. LiveData `Transformations`

Aplikacja GuessTheWord, nad którą pracowałeś w poprzednich trzech ćwiczeniach, implementuje wzorzec obserwatora LiveData do obserwowania danych ViewModel. Widoki w kontrolerze interfejsu użytkownika obserwują LiveData w ViewModel i aktualizują wyświetlane dane.
Podczas przekazywania LiveData między komponentami czasami możesz chcieć zmapować lub przekształcić dane. Kod może wymagać wykonania obliczeń, wyświetlenia tylko podzbioru danych lub zmiany sposobu ich wyświetlania. Na przykład dla słowa LiveData można utworzyć transformację, która zwraca liczbę liter w słowie zamiast samego słowa.

Można przekształcić LiveData przy użyciu metod pomocniczych w klasie `Transformations`:

- `Transformations.map`
- `Transformations.switchMap`

W tym ćwiczeniu dodasz odliczanie czasu w aplikacji. Dowiesz się, jak korzystać z `Transformations.map()` na `LiveData` aby przekształcić upływający czas w format wyświetlany na ekranie.

# Czego się nauczysz

- Jak korzystać z `Transformations` z `LiveData`

# Co będziesz robić

- Dodaj licznik czasu, aby zakończyć grę.
- Użyj `Transformations.map()` aby przekształcić jedną `LiveData` w drugą.

# 4. 4. Zadanie: dodaj licznik czasu

W tym zadaniu dodajesz minutnik do aplikacji. Zamiast końca gry, gdy lista słów jest pusta, gra kończy się po upływie czasu. Android zapewnia klasę narzędzi o nazwie `CountDownTimer` , której używasz do implementacji timera.

Dodaj logikę timera do `GameViewModel` , aby zegar nie został zniszczony podczas zmian konfiguracji. Fragment zawiera kod aktualizujący widok tekstu timera w miarę upływu czasu.

Wykonaj następujące kroki w klasie `GameViewModel`:

1. Utwórz obiekt **`companion`** do przechowywania stałych timera..

```
companion object {

    // Time when the game is over
    private const val DONE = 0L

    // Countdown time interval
    private const val ONE_SECOND = 1000L
```

```
    // Total time for the game
    private const val COUNTDOWN_TIME = 60000L

}
```

2. Aby zapisać czas odliczania timera, dodaj zmienną `MutableLiveData` o nazwie **`_currentTime`** oraz backing property, **`currentTime`**.

```
// Countdown time
private val _currentTime = MutableLiveData<Long>()
val currentTime: LiveData<Long>
    get() = _currentTime
```

3. Dodaj prywatną zmienną składową o nazwie **`timer`** typu `CountDownTimer`. Błąd inicjalizacji zostanie rozwiązany w następnym kroku.

```
private val timer: CountDownTimer
```

4. W bloku `init` block, zainicjuj i uruchom stoper. Podaj całkowity czas, **`COUNTDOWN_TIME`**. Dla przedziału czasowego użyj, **`ONE_SECOND`**. Zastąp metody wywołania zwrotnego `onTick()` i `onFinish()` i uruchom licznik czasu.

```
// Creates a timer which triggers the end of the game when it finishes
timer = object : CountDownTimer(COUNTDOWN_TIME, ONE_SECOND) {

    override fun onTick(millisUntilFinished: Long) {

    }

    override fun onFinish() {

    }
}

timer.start()
```

5. Zaimplementuj metodę wywołania zwrotnego `onTick()` która jest wywoływana w każdym interwale lub w każdym tiku. Zaktualizuj **`_currentTime`**, , używając przekazanego parametru `millisUntilFinished`. `millisUntilFinished` to czas w milisekundach, po upływie którego licznik czasu kończy się. Konwertuj `millisUntilFinished` na sekundy i przypisz go do **`_currentTime`**.

```
override fun onTick(millisUntilFinished: Long)
{
    _currentTime.value = millisUntilFinished/ONE_SECOND
}
```

6. Metoda wywołania zwrotnego `onFinish()` jest wywoływana po zakończeniu timera. Zaimplementuj `onFinish()` aby zaktualizować **`_currentTime`** i wywołać zdarzenie zakończenia gry.

```
override fun onFinish() {
    _currentTime.value = DONE
    onGameFinish()
```

```
}
```

7. Zaktualizuj metodę `nextWord()` aby zresetować listę słów, gdy lista jest pusta, zamiast kończyć grę.

```
private fun nextWord() {
    // Shuffle the word list, if the list is empty
    if (wordList.isEmpty()) {
        resetList()
    } else {
    // Remove a word from the list
    _word.value = wordList.removeAt(0)
}
```

8. W metodzie `onCleared()` anuluj licznik czasu, aby uniknąć wycieków pamięci. Możesz usunąć instrukcję dziennika, ponieważ nie jest już potrzebna. Metoda `onCleared()` jest wywoływana przed zniszczeniem `ViewModel`.

```
override fun onCleared() {
    super.onCleared()
    // Cancel the timer
    timer.cancel()
}
```

9. Uruchom aplikację i zagraj w grę. Poczekaj 60 sekund, a gra zakończy się automatycznie. Tekst timera nie jest jednak wyświetlany na ekranie.

# 5. Zadanie: dodaj transformację do LiveData

Metoda `Transformations.map()` zapewnia sposób wykonywania manipulacji danymi na źródłowym `LiveData` i zwrócenia wynikowego `LiveData`. Transformacje te nie są obliczane, chyba że obserwator obserwuje zwrócony obiekt `LiveData`.

Ta metoda przyjmuje źródło `LiveData` i funkcję jako parametry. Funkcja manipuluje źródłowym `LiveData`.

**Uwaga:** Funkcja lambda przekazana do `Transformation.map()` jest wykonywana w głównym wątku, więc nie uwzględniaj długotrwałych zadań.

W tym zadaniu formatujesz obiekt `LiveData` upływającego czasu w nowy ciąg Obiekt LiveData w formacie „MM: SS”. Sformatowany czas, który upłynął, jest również wyświetlany na ekranie.

Plik układu `game_fragment.xml` zawiera już widok tekstowy timera. Do tej pory widok tekstu nie miał tekstu do wyświetlenia, więc tekst timera nie był widoczny.

1. W klasie `GameViewModel` class, po utworzeniu wystąpienia **`currentTime`**, utwórz nowy obiekt `LiveData` object o nazwie `currentTimeString`. Ten obiekt jest przeznaczony do sformatowanej wersji ciągu `currentTime`.
2. Użyj `Transformations.map()` aby zdefiniować `currentTimeString`. Przekaż `currentTime` i funkcję lambda, aby sformatować czas. Możesz zaimplementować funkcję lambda za pomocą metody narzędzia `DateUtils.formatElapsedTime()`

utility method, która przyjmuje `long` liczba milisekund i formatuje ją do formatu ciągu „MM: SS”.

```
// The String version of the current time
val currentTimeString = Transformations.map(currentTime) { time ->
    DateUtils.formatElapsedTime(time)
}
```

3. W pliku `game_fragment.xml` w widoku timera, powiąż atrybut `text` attribute z `currentTimeString` z `gameViewModel`.

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

4. Uruchom aplikację i zagraj w grę. Tekst timera aktualizuje się co sekundę. Zauważ, że gra nie kończy się po przejściu wszystkich słów. Gra kończy się teraz, gdy upłynie czas.