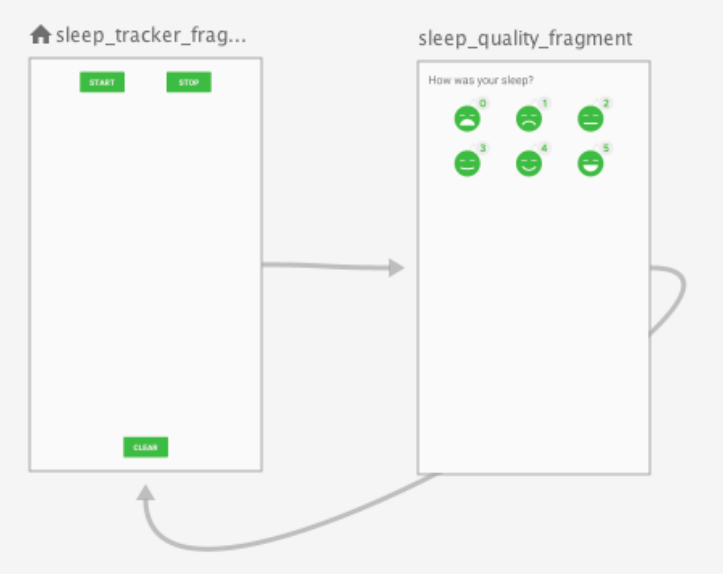
3. Zadanie: Dodaj nawigację

To ćwiczenie zakłada, że wiesz, jak zaimplementować nawigację za pomocą fragmentów i pliku nawigacji. Aby zaoszczędzić pracę, zapewniono sporo tego kodu.

Step 1: Sprawdź kod

1. Kontynuuj z kodem z końca ostatniego ćwiczenia.
2. W kodzie sprawdź`SleepQualityFragment`. Ta klasa nadmuchuje układ, pobiera aplikację i zwraca `binding.root`.
3. Otwórz **navigation.xml** w edytorze projektu. Widzisz, że istnieje ścieżka nawigacyjna od `SleepTrackerFragment` do `SleepQualityFragment`, iz powrotem od `SleepQualityFragment` do `SleepTrackerFragment`.



4. Sprawdź kod dla pliku**navigation.xml**. W szczególności poszukaj`<argument>` o nazwie `sleepNightKey`**.**

   Gdy użytkownik przejdzie z `SleepTrackerFragment` do `SleepQualityFragment`, , aplikacja przekaże `sleepNightKey` do `SleepQualityFragment` noc, która wymaga aktualizacji.

Step 2: Dodaj nawigację do śledzenia jakości snu

Wykres nawigacyjny zawiera już ścieżki od `SleepTrackerFragment` do `SleepQualityFragment` iz powrotem. Jednak procedury obsługi kliknięć, które implementują nawigację z jednego fragmentu do drugiego, nie są jeszcze zakodowane. Dodajesz teraz ten kod w `ViewModel`.

W module obsługi kliknięć ustawiasz `LiveData` który zmienia się, gdy aplikacja ma nawigować do innego miejsca docelowego. Fragment obserwuje ten `LiveData`. Kiedy dane się zmieniają, fragment nawiguje do miejsca docelowego i informuje model widoku, że to zrobione, co resetuje zmienną stanu.

1. Otwórz `SleepTrackerViewModel`. Musisz dodać nawigację, aby po dotknięciu przycisku **Stop** aplikacja przechodzi do `SleepQualityFragment` w celu zebrania oceny jakości.
2. W `SleepTrackerViewModel`, utwórz `LiveData` który zmienia się, gdy chcesz, aby aplikacja przechodziła do `SleepQualityFragment`. Użyj enkapsulacji, aby ujawnić tylko wersję `LiveData` dostępną dla `ViewModel`.

   Możesz umieścić ten kod w dowolnym miejscu na najwyższym poziomie ciała klasy.

```
private val _navigateToSleepQuality = MutableLiveData<SleepNight>()

val navigateToSleepQuality: LiveData<SleepNight>
   get() = _navigateToSleepQuality
```

3. Dodaj funkcję `doneNavigating()`, która resetuje zmienną uruchamiającą nawigację.

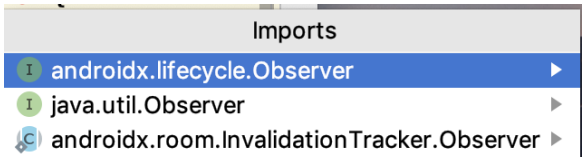
```
fun doneNavigating() {
   _navigateToSleepQuality.value = null
}
```

4. W module obsługi kliknięć przycisku **Stop**, `onStopTracking()`,uruchom nawigację do `SleepQualityFragment`. Ustaw zmienną `_navigateToSleepQuality` na końcu funkcji jako ostatnią rzecz w bloku uruchomienia `launch{}` Zauważ, że ta zmienna jest ustawiona na `night`. Gdy ta zmienna ma wartość, aplikacja przechodzi do `SleepQualityFragment`.

```
_navigateToSleepQuality.value = oldNight
```

5. `SleepTrackerFragment` obserwpwać `_navigateToSleepQuality` aby aplikacja wiedziała, kiedy nawigować. W `SleepTrackerFragment`, w `onCreateView()`,dodaj obserwatora do `navigateToSleepQuality()`.Pamiętaj, że importowanie tego jest niejednoznaczne i musisz zaimportować `androidx.lifecycle.Observer`.

```
sleepTrackerViewModel.navigateToSleepQuality.observe(this, Observer {
})
```



6. W bloku obserwatora nawiguj i podaj identyfikator bieżącej nocy, a następnie wywołaj `doneNavigating()`.Jeśli import jest niejednoznaczny, zaimportuj `androidx.navigation.fragment.findNavController`.

```
night ->
night?.let {
   this.findNavController().navigate(
           SleepTrackerFragmentDirections
```

```
.actionSleepTrackerFragmentToSleepQualityFragment(night.nightId))
   sleepTrackerViewModel.doneNavigating()
}
```

7. Zbuduj i uruchom aplikację. Stuknij **Start**, a następnie **Stop**, aby przejść do ekranu `SleepQualityFragment` Aby wrócić, użyj systemowego przycisku Wstecz.

4. Zadanie: Zapisz jakość snu

W tym zadaniu rejestrujesz jakość snu i wracasz do fragmentu modułu snu. Wyświetlacz powinien zaktualizować się automatycznie, aby pokazać zaktualizowaną wartość użytkownikowi. Musisz utworzyć `ViewModel` i `ViewModelFactory`, oraz zaktualizować `SleepQualityFragment`.

Step 1: Utwórz ViewModel i ViewModelFactory

1. W pakiecie `sleepquality` utwórz lub otwórz **SleepQualityViewModel.kt.**
2. Utwórz klasę `SleepQualityViewModel` , która przyjmuje parametry `sleepNightKey` i bazę danych jako argumenty. Podobnie jak w przypadku `SleepTrackerViewModel`, musisz przekazać `database` z fabryki. Musisz także przejść w tryb `sleepNightKey` z nawigacji.

```
class SleepQualityViewModel(
       private val sleepNightKey: Long = 0L,
       val database: SleepDatabaseDao) : ViewModel() {
}
```

3. W klasie `SleepQualityViewModel` zdefiniuj `Job` i `uiScope`, i przesłoń `onCleared()`.

```
private val viewModelJob = Job()
private val uiScope = CoroutineScope(Dispatchers.Main + viewModelJob)

override fun onCleared() {
   super.onCleared()
   viewModelJob.cancel()
}
```

4. Aby wrócić do `SleepTrackerFragment` przy użyciu tego samego wzorca jak powyżej, zadeklaruj `_navigateToSleepTracker`. Zaimplementuj `navigateToSleepTracker` i `doneNavigating()`.

```
private val _navigateToSleepTracker = MutableLiveData<Boolean?>()

val navigateToSleepTracker: LiveData<Boolean?>
   get() = _navigateToSleepTracker

fun doneNavigating() {
   _navigateToSleepTracker.value = null
}
```

5. Utwórz moduł obsługi jednego kliknięcia, `onSetSleepQuality()`,dla wszystkich obrazów o jakości uśpienia.

Użyj tego samego wzoru coroutine, jak w poprzednim ćwiczeniu:

- Uruchom coroutine w `uiScope`, i przejdź do dyspozytora we / wy.
- Pobierz `tonight` wieczorem za pomocą `sleepNightKey`.
- Ustaw jakość snu.
- Zaktualizuj bazę danych.
- Wywołaj nawigacje.

Zauważ, że poniższy przykład kodu wykonuje całą pracę w module obsługi kliknięć, zamiast faktoryzować operację bazy danych w innym kontekście.

```
fun onSetSleepQuality(quality: Int) {
        uiScope.launch {
            // IO is a thread pool for running operations that access the
disk, such as
            // our Room database.
            withContext(Dispatchers.IO) {
                val tonight = database.get(sleepNightKey) ?:
return@withContext
                tonight.sleepQuality = quality
                database.update(tonight)
            }

            // Setting this state variable to true will alert the observer
and trigger navigation.
            _navigateToSleepTracker.value = true
        }
    }
```

5. W pakiecie `sleepquality` utwórz lub otwórz `SleepQualityViewModelFactory.kt` i dodaj klasę `SleepQualityViewModelFactory` jak pokazano poniżej. Ta klasa używa wersji tego samego kodu, który widziałeś wcześniej. Sprawdź kod, zanim przejdziesz dalej.

```
class SleepQualityViewModelFactory(
       private val sleepNightKey: Long,
       private val dataSource: SleepDatabaseDao) :
ViewModelProvider.Factory {
   @Suppress("unchecked_cast")
   override fun <T : ViewModel?> create(modelClass: Class<T>): T {
       if (modelClass.isAssignableFrom(SleepQualityViewModel::class.java))
{
           return SleepQualityViewModel(sleepNightKey, dataSource) as T
       }
       throw IllegalArgumentException("Unknown ViewModel class")
   }
}
```

Step 2: Zaktualizuj SleepQualityFragment

1. Otwórz `SleepQualityFragment.kt`.
2. W `onCreateView()`,po otrzymaniu `application`, musisz uzyskać `arguments` dostarczone z nawigacją. Te argumenty znajdują się w `SleepQualityFragmentArgs`. Musisz wyodrębnić je z pakietu (kundle).

```
val arguments = SleepQualityFragmentArgs.fromBundle(arguments!!)
```

3. Następnie pobierz `dataSource`.

```
val dataSource = SleepDatabase.getInstance(application).sleepDatabaseDao
```

4. Utwórz fabrykę, przekazując `dataSource` i `sleepNightKey`.

```
val viewModelFactory =
SleepQualityViewModelFactory(arguments.sleepNightKey, dataSource)
```

5. Uzyskaj odwołanie do `ViewModel`.

```
val sleepQualityViewModel =
       ViewModelProviders.of(
               this,
viewModelFactory).get(SleepQualityViewModel::class.java)
```

6. Dodaj `ViewModel` do obiektu powiązania. (Jeśli zobaczysz błąd w obiekcie wiązania, zignoruj go na razie).

```
binding.sleepQualityViewModel = sleepQualityViewModel
```

7. Dodaj obserwatora. Po wyświetleniu monitu zaimportuj `androidx.lifecycle.Observer`.

```
sleepQualityViewModel.navigateToSleepTracker.observe(this, Observer {
   if (it == true) { // Observed state is true.
       this.findNavController().navigate(

SleepQualityFragmentDirections.actionSleepQualityFragmentToSleepTrackerFrag
ment())
       sleepQualityViewModel.doneNavigating()
   }
})
```

Krok 3: Zaktualizuj plik układu i uruchom aplikację

1. Otwórz plik układu `fragment_sleep_quality.xml` W bloku `<data>` dodaj zmienną dla `SleepQualityViewModel`.

```
 <data>
       <variable
           name="sleepQualityViewModel"

type="com.example.android.trackmysleepquality.sleepquality.SleepQualityView
Model" />
   </data>
```

2. Dla każdego z sześciu obrazów o jakości snu dodaj moduł obsługi kliknięć, taki jak ten poniżej. Dopasuj ocenę jakości snu do obrazu..

```
android:onClick="@{() -> sleepQualityViewModel.onSetSleepQuality(5)}"
```

3. Oczyść i odbuduj swój projekt. To powinno rozwiązać wszelkie błędy związane z obiektem wiązania. W przeciwnym razie wyczyść pamięć podręczną (**File > Invalidate Caches / Restart**) i odbuduj aplikację.

Przed kontynuowaniem upewnij się, że aplikacja działa bez błędów i że możesz teraz nagrywać jakość snu.

Właśnie zbudowałeś kompletną aplikację bazy danych Room za pomocą coroutines.

5. Zadanie: kontroluj widoczność przycisków i dodaj pasek

Teraz Twoja aplikacja działa. Użytkownik może dotknąć Start i Stop tyle razy, ile chce. Gdy użytkownik stuknie Stop, może wprowadzić jakość snu. Gdy użytkownik stuknie opcję Wyczyść, wszystkie dane są usuwane po cichu w tle. Jednak wszystkie przyciski są zawsze włączone i klikalne, co nie psuje aplikacji, ale pozwala użytkownikom tworzyć niepełne noce snu.

W tym ostatnim zadaniu nauczysz się korzystać z map transformacji do zarządzania widocznością przycisków, aby użytkownicy mogli tylko dokonać właściwego wyboru. Możesz użyć podobnej metody, aby wyświetlić przyjazny komunikat po usunięciu wszystkich danych.

Step 1: Zaktualizuj stany przycisków

Chodzi o to, aby ustawić stan przycisku tak, aby na początku był włączony tylko przycisk Start, co oznacza, że można go kliknąć.

Gdy użytkownik stuknie Start, przycisk Stop zostanie włączony, a Start nie. Przycisk Wyczyść jest aktywny tylko wtedy, gdy w bazie danych znajdują się dane.

1. Otwórz plik układu `fragment_sleep_tracker.xml`.
2. Dodaj właściwość `android:enabled` do każdego przycisku. `android:enabled` to wartość logiczna, która wskazuje, czy przycisk jest włączony. (Można kliknąć przycisk włączony, przycisk wyłączony nie.)

```
start_button:

android:enabled="@{sleepTrackerViewModel.startButtonVisible}"

stop_button:

android:enabled="@{sleepTrackerViewModel.stopButtonVisible}"

clear_button:

android:enabled="@{sleepTrackerViewModel.clearButtonVisible}"
```

3. Otwórz `SleepTrackerViewModel` i utwórz trzy odpowiednie zmienne. Przypisz każdej zmiennej transformację, która ją testuje.

- Przycisk **Start** powinien być włączony, gdy `tonight` jest `null`.
- Przycisk **Stop** powinien być włączony, gdy `tonight` nie jest `null`.
- Przycisk **Clear** Wyczyść powinien być włączony tylko wtedy, `nights`, a tym samym baza danych, zawierają dane.

```
val startButtonVisible = Transformations.map(tonight) {
   it == null
}
val stopButtonVisible = Transformations.map(tonight) {
   it != null
}
val clearButtonVisible = Transformations.map(nights) {
   it?.isNotEmpty()
}
```

4. Uruchom aplikację i eksperymentuj z przyciskami.

Tip: Ustawianie wyglądu wyłączonego widoku

`enabled` trybut nie jest taki sam jak atrybut `visibility`. `enabled` atrybut określa tylko, czy widok jest włączony, a nie czy widok jest widoczny.

Znaczenie „włączone" różni się w zależności od podklasy. Użytkownik może edytować tekst w aktywnym EditText, ale nie w wyłączonym EditText. Użytkownik może dotknąć włączonego przycisku, ale nie może go wyłączyć.

Domyślny styl jest stosowany do wyłączonego Widoku, aby wizualnie reprezentować, że Widok nie jest aktywny.

Jeśli jednak widok ma atrybut tła lub atrybut textColor, wartości tych atrybutów są używane podczas wyświetlania widoku, nawet jeśli widok jest wyłączony.

Aby zdefiniować, które kolory mają być używane dla stanów włączonych i wyłączonych, użyj `ColorStateList` dla koloru tekstu i `StateListDrawable` dla koloru tła.

Step 2: Użyj paska snackbar aby powiadomić użytkownika

Gdy użytkownik wyczyści bazę danych, pokaż mu potwierdzenie za pomocą widżetu `Snackbar` Pasek *snackbar* przekazuje krótką informację zwrotną na temat operacji za pośrednictwem komunikatu na dole ekranu. Pasek przekąsek znika po upływie limitu czasu, po interakcji użytkownika w innym miejscu na ekranie lub po przesunięciu go poza ekran.

Pokazywanie paska snackbar jest zadaniem interfejsu użytkownika i powinno się zdarzyć we fragmencie. Decyzja o wyświetleniu paska snackbar odbywa się w `ViewModel`. Aby skonfigurować i uruchomić pasek snackbar po wyczyszczeniu danych, możesz użyć tej samej techniki, co w przypadku wyzwalania nawigacji.

1. W `SleepTrackerViewModel`, utwórz enkapsulowane zdarzenie.

```
private var _showSnackbarEvent = MutableLiveData<Boolean>()

val showSnackBarEvent: LiveData<Boolean>
   get() = _showSnackbarEvent
```

2. Następnie zaimplementuj `doneShowingSnackbar()`.

```
fun doneShowingSnackbar() {
   _showSnackbarEvent.value = false
}
```

3. W `SleepTrackerFragment`, w `onCreateView()`,dodaj obserwatora:

```
sleepTrackerViewModel.showSnackBarEvent.observe(this, Observer { })
```

4. W bloku obserwatora wyświetl pasek snackbar i natychmiast zresetuj zdarzenie.

```
   if (it == true) { // Observed state is true.
       Snackbar.make(
               activity!!.findViewById(android.R.id.content),
               getString(R.string.cleared_message),
               Snackbar.LENGTH_SHORT // How long to display the message.
       ).show()
       sleepTrackerViewModel.doneShowingSnackbar()
   }
```

5. W `SleepTrackerViewModel`, , uruchom zdarzenie w metodzie `onClear()`ustaw wartość zdarzenia na `true` w bloku `launch`:

```
_showSnackbarEvent.value = true
```

6. Zbuduj i uruchom aplikację!