Część 3: Nawigacja

Zwiększ możliwości użytkowników nawigowania po różnych ekranach w aplikacji, do nich i z powrotem, aby zapewnić spójne i przewidywalne środowisko użytkownika.

Spis treści

Nawiguj między ekranami ... 7
Kolekcje w Kotlinie ... 7
1. Zanim zaczniesz ... 7
2. Poznaj kolekcje ... 7
3. Praca z kolekcjami ... 11
4. Dowiedz się o lambdach i funkcjach wyższego rzędu ... 13
5. Twórz listy słów ... 17
6. Podsumowanie ... 19
Działania i zamiary ... 20
1. Wstęp ... 20
2. Kod startowy ... 20
3. Przegląd aplikacji Words ... 23
4. Wprowadzenie do intencji ... 24
5. Skonfiguruj wyraźną intencję ... 26
6. Skonfiguruj aktywność szczegółów ... 27
7. Stwórz ukrytą intencję ... 30
8. Skonfiguruj menu i ikony ... 32
9. Przycisk Menu Wdrażania ... 35
10. Kod rozwiązania ... 37
12. Dowiedz się więcej ... 39
Etapy cyklu życia działalności ... 40
1. Witamy! ... 40
2. Przegląd aplikacji ... 40
3. Poznaj metody cyklu życia i dodaj podstawowe rejestrowanie ... 43
4. Poznaj przypadki użycia w cyklu życia ... 50
5. Poznaj zmiany w konfiguracji ... 56
6. Podsumowanie ... 61
7. Dowiedz się więcej ... 62
Wprowadzenie do komponentu Nawigacja ... 63
Fragmenty i komponent nawigacyjny ... 63
1. Zanim zaczniesz ... 63
2. Kod startowy ... 65

3. Fragmenty i cykl życia fragmentów ... 67
4. Utwórz pliki fragmentów i układów ... 68
5. Implementuj fragment listy listów ... 71
6. Konwertuj DetailActivity na WordListFragment ... 75
7. Komponent nawigacji Jetpack ... 78
8. Korzystanie z wykresu nawigacyjnego ... 80
9. Pobieranie argumentów w WordListFragment ... 86
10. Zaktualizuj etykiety fragmentów ... 87
11. Kod rozwiązania ... 88
12. Podsumowanie ... 90
Przetestuj komponenty nawigacyjne ... 91
1. Zanim zaczniesz ... 91
2. Przegląd aplikacji startowej ... 93
3. Utwórz katalogi testów ... 93
4. Utwórz test oprzyrządowania Class ... 93
5. Dodaj niezbędne zależności ... 94
6. Napisz test komponentu nawigacyjnego ... 94
7. Kod rozwiązania ... 95
8. Unikaj powtarzania kodu z adnotacjami ... 95
9. Gratulacje ... 98
Elementy architektury ... 99
Przechowuj dane w ViewModel ... 99
1. Zanim zaczniesz ... 99
3. Dowiedz się więcej o architekturze aplikacji ... 104
4. Dodaj ViewModel ... 105
5. Przenieś dane do ViewModel ... 108
6. Cykl życia ViewModelu ... 110
7. Wypełnij ViewModel ... 112
8. Dialogi ... 115
9. Zaimplementuj OnClickListener dla przycisku Prześlij ... 118
10. Zaimplementuj przycisk Pomiń ... 122
11. Sprawdź, czy ViewModel zachowuje dane ... 123
12. Zaktualizuj logikę restartu gry ... 124
13. Kod rozwiązania ... 126

14. Podsumowanie 131
15. Dowiedz się więcej 131
Użyj LiveData z ViewModel 133
1. Zanim zaczniesz 133
2. Przegląd aplikacji startowej 134
3. Co to są dane na żywo 134
4. Dodaj LiveData do bieżącego zaszyfrowanego słowa 135
5. Dołącz obserwatora do obiektu LiveData 136
6. Dołącz obserwatora do punktacji i liczby słów 137
7. Użyj LiveData z wiązaniem danych 140
8. Dodaj zmienne wiążące dane 143
10. Przetestuj aplikację Unscramble z włączoną funkcją Talkback 147
11. Usuń nieużywany kod 148
12. Kod rozwiązania 148
13. Podsumowanie 150
14. Dowiedz się więcej 150
Przykłady zaawansowanych aplikacji nawigacyjnych 151
Wspólny model widoku we fragmentach 151
1. Zanim zaczniesz 151
2. Przegląd aplikacji startowej 152
3. Uzupełnij wykres nawigacji 155
4. Utwórz udostępniony ViewModel 163
5. Użyj ViewModel, aby zaktualizować interfejs użytkownika 166
6. Użyj ViewModel z wiązaniem danych 168
7. Zaktualizuj fragment odbioru i podsumowania, aby użyć modelu widoku 172
8. Oblicz cenę ze szczegółów zamówienia 179
9. Skonfiguruj odbiorniki kliknięcia za pomocą wiązania słuchacza 189
10. Kod rozwiązania 193
11. Podsumowanie 195
12. Dowiedz się więcej 195
Nawigacja i tylny stos 196
1. Zanim zaczniesz 196
2. Przegląd aplikacji startowej 197
3. Zaimplementuj zachowanie przycisku w górę 200

4. Dowiedz się o zadaniach i stosie wstecznym 201
5. Wyślij zamówienie 219
6. Kod rozwiązania 225
7. Podsumowanie 226
8. Dowiedz się więcej 227
9. Ćwicz na własną rękę 227
Testuj modele widoków i dane na żywo 229
1. Zanim zaczniesz 229
2. Przegląd aplikacji startowej 231
3. Utwórz katalogi testów jednostkowych 232
4. Utwórz klasę testów jednostkowych 232
5. Dodaj niezbędne zależności 232
6. Napisz test ViewModel 232
7. Kod rozwiązania 235
8. Gratulacje 235
Układy adaptacyjne 236
Układy adaptacyjne 236
1. Zanim zaczniesz 236
2. Obejrzyj wideo z kodem (opcjonalnie) 238
3. Przegląd aplikacji startowej 239
4. Wzór listy-szczegółów 244
5. Wzór układu przesuwnego panelu 246
6. Dodaj zależności biblioteczne 249
7. Skonfiguruj fragment listy sportowej xml 249
8. Zamień okienko szczegółów 253
9. Dodaj niestandardową nawigację wsteczną 255
10. Tryb blokady 261
11. Kod rozwiązania 262
12. Dowiedz się więcej 263
Projekt:Aplikacja taca na lunch 264
1. Zanim zaczniesz 264
2. Zakończony przegląd aplikacji 265
3. Rozpocznij 268
4. Przetestuj swoją aplikację 277

5. Opcjonalnie: przekaż nam swoją opinię!.. 278

Nawiguj między ekranami

Dodaj kolejny ekran do aplikacji, dodając drugą czynność i użyj intencji, aby przejść do niej. Naucz się również podstaw cyklu życia działania, gdy przechodzisz do różnych działań i z nich wychodzisz.

Kolekcje w Kotlinie

1. Zanim zaczniesz

W tym laboratorium dowiesz się więcej o kolekcjach, lambdach i funkcjach wyższego rzędu w Kotlinie.

Warunki wstępne

- Podstawowe zrozumienie pojęć Kotlina przedstawione w poprzednich ćwiczeniach z programowania.
- Zaznajomiony z wykorzystaniem Kotlin Playground do tworzenia i edycji programów Kotlin.

Czego się nauczysz

- Jak pracować z kolekcjami, w tym zestawami i mapami
- Podstawy lambd
- Podstawy funkcji wyższego rzędu

Czego potrzebujesz

- Komputer z połączeniem internetowym umożliwiającym dostęp do Placu Zabaw Kotlin

2. Poznaj kolekcje

Kolekcja to *grupa* powiązanych elementów, takich jak lista słów lub zestaw rekordów pracowników. Kolekcja może zawierać elementy uporządkowane lub nieuporządkowane, a elementy mogą być unikatowe lub nie. Poznałeś już jeden rodzaj kolekcji, listy. Listy mają kolejność pozycji, ale pozycje nie muszą być unikatowe.

Podobnie jak w przypadku list, Kotlin rozróżnia kolekcje mutowalne i niezmienne. Kotlin udostępnia wiele funkcji do dodawania lub usuwania elementów, przeglądania i manipulowania kolekcjami.

Stwórz listę

W tym zadaniu zapoznasz się z tworzeniem listy liczb i ich sortowaniem.

1. Otwórz plac zabaw Kotlin .
2. Zastąp dowolny kod tym kodem:

```
fun main() {
    val numbers = listOf(0, 3, 8, 4, 0, 5, 5, 8, 9, 2)
    println("list:   ${numbers}")
}
```

3. Uruchom program, dotykając zielonej strzałki i spójrz na wyświetlane wyniki:

```
list:   [0, 3, 8, 4, 0, 5, 5, 8, 9, 2]
```

4. Lista zawiera 10 liczb od 0 do 9. Niektóre liczby pojawiają się więcej niż jeden raz, a niektóre nie pojawiają się wcale.
5. Kolejność elementów na liście ma znaczenie: pierwsza pozycja to `0`, druga pozycja to `3`i tak dalej. Elementy pozostaną w tej kolejności, chyba że je zmienisz.
6. Przypomnij sobie z wcześniejszych ćwiczeń z programowania, że listy mają wiele wbudowanych funkcji, takich jak `sorted()`, które zwracają kopię listy posortowaną w kolejności rosnącej. Po `println()`, dodaj wiersz do programu, aby wydrukować posortowaną kopię listy:

```
println("sorted: ${numbers.sorted()}")
```

7. Uruchom program ponownie i spójrz na wyniki:

```
list:   [0, 3, 8, 4, 0, 5, 5, 8, 9, 2]
sorted: [0, 0, 2, 3, 4, 5, 5, 8, 8, 9]
```

Dzięki posortowaniu liczb łatwiej jest sprawdzić, ile razy każdy numer pojawia się na liście lub czy w ogóle się nie pojawia.

Dowiedz się o zestawach

Innym rodzajem kolekcji w Kotlinie jest *zestaw* . Jest to grupa powiązanych elementów, ale w przeciwieństwie do listy nie może być żadnych duplikatów, a kolejność nie ma znaczenia. Element może być w zestawie lub nie, ale jeśli jest w zestawie, to jest tylko jeden jego egzemplarz. Jest to podobne do matematycznej koncepcji zbioru. Na przykład jest zestaw książek, które przeczytałeś. Wielokrotne czytanie książki nie zmienia faktu, że znajduje się ona w zestawie książek, które przeczytałeś.

1. Dodaj te wiersze do swojego programu, aby przekonwertować listę na zestaw:

```
val setOfNumbers = numbers.toSet()
println("set:    ${setOfNumbers}")
```

2. Uruchom swój program i spójrz na wyniki:

```
list:   [0, 3, 8, 4, 0, 5, 5, 8, 9, 2]
sorted: [0, 0, 2, 3, 4, 5, 5, 8, 8, 9]
set:    [0, 3, 8, 4, 5, 9, 2]
```

Wynik zawiera wszystkie liczby z oryginalnej listy, ale każdy pojawia się tylko raz. Zwróć uwagę, że są one w tej samej kolejności, co na oryginalnej liście, ale ta kolejność nie ma znaczenia dla zestawu.

3. Zdefiniuj zestaw zmienny i niezmienny i zainicjuj je tym samym zestawem liczb, ale w innej kolejności, dodając te wiersze:

```
val set1 = setOf(1,2,3)
val set2 = mutableSetOf(3,2,1)
```

4. Dodaj wiersz do wydrukowania, czy są równe:

```
println("$set1 == $set2: ${set1 == set2}")
```

5. Uruchom swój program i spójrz na nowe wyniki:

```
[1, 2, 3] == [3, 2, 1]: true
```

Nawet jeśli jeden jest zmienny, a drugi nie, i mają elementy w innej kolejności, są one uważane za równe, ponieważ zawierają dokładnie ten sam zestaw elementów.

Jedną z głównych operacji, które możesz wykonać na zestawie, jest sprawdzenie, czy dana pozycja znajduje się w zestawie, czy nie contains(). Widziałeś już contains()wcześniej, ale użyłeś go na liście.

6. Dodaj tę linię do swojego programu, aby wydrukować, jeśli 7jest w zestawie:

```
println("contains 7: ${setOfNumbers.contains(7)}")
```

7. Uruchom swój program i spójrz na dodatkowe wyniki:

```
contains 7: false
```

Możesz spróbować przetestować go również z wartością, która jest w zestawie.

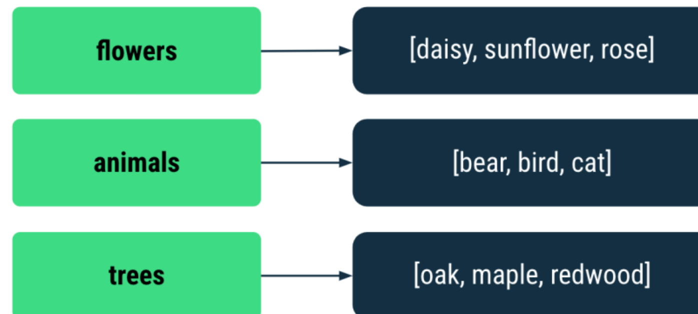
All of the code above:

```
fun main() {
    val numbers = listOf(0, 3, 8, 4, 0, 5, 5, 8, 9, 2)
    println("list:   ${numbers}")
    println("sorted: ${numbers.sorted()}")
    val setOfNumbers = numbers.toSet()
    println("set:    ${setOfNumbers}")
    val set1 = setOf(1,2,3)
    val set2 = mutableSetOf(3,2,1)
    println("$set1 == $set2: ${set1 == set2}")
    println("contains 7: ${setOfNumbers.contains(7)}")
}
```

Podobnie jak w przypadku zbiorów matematycznych, w Kotlinie można również wykonywać operacje takie jak przecięcie (∩) lub suma (∪) dwóch zbiorów, używając intersect()lub union().

Dowiedz się więcej o mapach

Ostatnim typem kolekcji, o którym nauczysz się podczas tego ćwiczenia z programowania, jest *mapa* lub *słownik* . Mapa to zestaw *par klucz-wartość* , który ma ułatwić wyszukiwanie wartości przy danym kluczu. Klucze są unikatowe, a każdy klucz jest mapowany na dokładnie jedną wartość, ale wartości mogą mieć duplikaty. Wartości na mapie mogą być ciągami, liczbami lub obiektami — nawet inną kolekcją, taką jak lista lub zestaw.



Mapa jest przydatna, gdy masz pary danych i możesz zidentyfikować każdą parę na podstawie jej klucza. Klucz „odwzorowuje" odpowiednią wartość.

1. Na placu zabaw Kotlin zastąp cały kod tym kodem, który tworzy zmienną mapę do przechowywania imion osób i ich wieku:

```
fun main() {
   val peopleAges = mutableMapOf<String, Int>(
      "Fred" to 30,
      "Ann" to 23
   )
   println(peopleAges)
}
```

Tworzy to zmienną mapę `String`(klucz) na `Int`(wartość), inicjuje mapę z dwoma wpisami i drukuje elementy.

2. Uruchom swój program i spójrz na wyniki:

```
{Fred=30, Ann=23}
```

3. Aby dodać więcej wpisów do mapy, możesz skorzystać z `put()`funkcji, podając klucz i wartość:

```
peopleAges.put("Barbara", 42)
```

4. Możesz także użyć notacji skróconej, aby dodać wpisy:

```
peopleAges["Joe"] = 51
```

Oto cały powyższy kod:

```
fun main() {
   val peopleAges = mutableMapOf<String, Int>(
      "Fred" to 30,
      "Ann" to 23
```

```
    )
    peopleAges.put("Barbara", 42)
    peopleAges["Joe"] = 51
    println(peopleAges)
}
```

5. Uruchom swój program i spójrz na wyniki:

   ```
   {Fred=30, Ann=23, Barbara=42, Joe=51}
   ```

   Jak wspomniano powyżej, klucze (nazwy) są unikalne, ale wartości (wiek) mogą mieć duplikaty. Jak myślisz, co się stanie, jeśli spróbujesz dodać przedmiot za pomocą jednego z tych samych kluczy?

6. Przed `println()`, dodaj ten wiersz kodu:

   ```
   peopleAges["Fred"] = 31
   ```

7. Uruchom swój program i spójrz na wyniki:

   ```
   {Fred=31, Ann=23, Barbara=42, Joe=51}
   ```

   Klucz `"Fred"`nie zostanie ponownie dodany, ale wartość, na którą jest mapowany, zostanie zaktualizowana do `31`.

   Jak widać, mapy są przydatne jako szybki sposób mapowania kluczy na wartości w kodzie!

3. Praca z kolekcjami

Chociaż mają różne cechy, różne typy kolekcji mają wiele cech wspólnych. Jeśli są zmienne, możesz dodawać lub usuwać elementy. Możesz wyliczyć wszystkie elementy, znaleźć konkretny element, a czasem przekonwertować jeden typ kolekcji na inny. Zrobiłeś to wcześniej, kiedy przekonwertowałeś a `List`na `Set`z `toSet()`. Oto kilka przydatnych funkcji do pracy z kolekcjami.

dla każdego

Załóżmy, że chcesz wydrukować elementy w programie `peopleAges`i podać imię i nazwisko oraz wiek osoby. Na przykład `"Fred is 31, Ann is 23,..."`i tak dalej. Nauczyłeś się o `for`pętlach we wcześniejszej lekcji programowania, więc możesz napisać pętlę za pomocą `for (people in peopleAges) { ... }`.

Jednak wyliczanie wszystkich obiektów w kolekcji jest powszechną operacją, więc Kotlin udostępnia `forEach()`, który przegląda wszystkie elementy za Ciebie i wykonuje operację na każdym z nich.

1. Na placu zabaw dodaj ten kod po `println()`:

   ```
   peopleAges.forEach { print("${it.key} is ${it.value}, ") }
   ```

   Jest podobny do `for`pętli, ale trochę bardziej zwarty. Zamiast określać zmienną dla bieżącego elementu, `forEach`używa specjalnego identyfikatora `it`.

Zauważ, że nie trzeba było dodawać nawiasów podczas wywoływania `forEach()`metody, po prostu przekaż kod w nawiasach klamrowych `{}`.

2. Uruchom swój program i spójrz na dodatkowe wyniki:

```
Fred is 31, Ann is 23, Barbara is 42, Joe is 51,
```

To bardzo zbliżone do tego, czego chcesz, ale na końcu jest dodatkowy przecinek.

Konwersja kolekcji na ciąg jest powszechną operacją, a dodatkowy separator na końcu również jest częstym problemem. Dowiesz się, jak sobie z tym poradzić w kolejnych krokach.

mapa

Funkcja map()(której nie należy mylić z powyższą kolekcją map lub słowników) stosuje przekształcenie do każdego elementu w kolekcji.

1. W swoim programie zastąp `forEach`instrukcję następującym wierszem:

```
println(peopleAges.map { "${it.key} is ${it.value}" }.joinToString(", ") )
```

2. Uruchom swój program i spójrz na dodatkowe wyniki:

```
Fred is 31, Ann is 23, Barbara is 42, Joe is 51
```

Ma poprawny wynik i bez dodatkowego przecinka! W jednym wierszu dużo się dzieje, więc przyjrzyj się temu bliżej.

- `peopleAges.map`stosuje transformację do każdego elementu w `peopleAges`i tworzy nową kolekcję przekształconych elementów
- Część w nawiasach klamrowych `{}`definiuje transformację do zastosowania do każdego elementu. Transformacja pobiera parę klucz-wartość i przekształca ją w ciąg, na przykład `<Fred, 31>`zamienia się w `Fred is 31`.
- `joinToString(", ")`dodaje każdy element w przekształconej kolekcji do ciągu, oddzielony przez `,`i wie, aby nie dodawać go do ostatniego elementu
- wszystko to jest połączone z `.`(operatorem kropki), tak jak robiłeś to z wywołaniami funkcji i dostępami do właściwości we wcześniejszych ćwiczeniach z programowania

filtr

Inną częstą operacją związaną z kolekcjami jest znalezienie elementów spełniających określony warunek. Funkcja filter()zwraca elementy w zgodnej kolekcji na podstawie wyrażenia.

1. Po `println()`, dodaj te wiersze:

```
val filteredNames = peopleAges.filter { it.key.length < 4 }
println(filteredNames)
```

Ponownie zauważ, że wywołanie `filter`to nie wymaga nawiasów i `it`odnosi się do bieżącego elementu na liście.

2. Uruchom swój program i spójrz na dodatkowe wyniki:

```
{Ann=23, Joe=51}
```

W tym przypadku wyrażenie pobiera długość klucza (a `String`) i sprawdza, czy jest ona mniejsza niż 4. Do nowej kolekcji dodawane są dowolne elementy, które pasują, to znaczy mają nazwę krótszą niż 4 znaki.

Typ zwracany po zastosowaniu filtru do mapy to nowa mapa (`LinkedHashMap`). Możesz wykonać dodatkowe przetwarzanie na mapie lub przekonwertować ją na inny typ kolekcji, np. listę.

4. Dowiedz się o lambdach i funkcjach wyższego rzędu

Lambdy

Czy wzorzec w powyższym kodzie wygląda znajomo?

```
peopleAges.forEach { print("${it.key} is ${it.value}") }
```

Istnieje zmienna (`peopleAges`) z wywołaną na niej funkcją (`forEach`). Zamiast nawiasów po nazwie funkcji z parametrami, zobaczysz kod w nawiasach klamrowych `{}`po nazwie funkcji. Ten sam wzorzec pojawia się w kodzie, który używa `map`i `filter`działa z poprzedniego kroku. Funkcja `forEach`zostaje wywołana na `peopleAges`zmiennej i używa kodu w nawiasach klamrowych.

To tak, jakbyś napisał małą funkcję w nawiasach klamrowych, ale nie ma nazwy funkcji. Ten pomysł — funkcja bez nazwy, która może być natychmiast użyta jako wyrażenie — jest naprawdę przydatnym pojęciem zwanym *wyrażeniem lambda* lub w skrócie lambda.

Prowadzi to do ważnego tematu, w jaki sposób możesz komunikować się z funkcjami w potężny sposób za pomocą Kotlina. Możesz przechowywać funkcje w zmiennych i klasach, przekazywać funkcje jako argumenty, a nawet zwracać funkcje. Można je traktować tak, jak zmienne innych typów, takie jak `Int`lub `String`.

Rodzaje funkcji

Aby umożliwić tego typu zachowanie, Kotlin ma coś, co nazywa się *typami funkcji* , gdzie można zdefiniować określony typ funkcji na podstawie jej parametrów wejściowych i wartości zwracanej. Występuje w następującym formacie:

Przykładowy typ funkcji:`(Int) -> Int`

Funkcja z powyższym typem funkcji musi przyjmować parametr typu `Int`i zwracać wartość typu `Int`. W notacji typu funkcji parametry są wymienione w nawiasach (oddzielone przecinkami, jeśli istnieje wiele parametrów). Następnie jest strzałka `->`, po której następuje typ zwracany.

Jaki rodzaj funkcji spełniałby te kryteria? Możesz mieć wyrażenie lambda, które potraja wartość wejściowej liczby całkowitej, jak pokazano poniżej. W przypadku składni wyrażenia lambda parametry są umieszczane jako pierwsze (podświetlone w czerwonym polu), po których następuje strzałka funkcji, a następnie treść funkcji (podświetlona w fioletowym polu). Ostatnie wyrażenie w lambdzie to wartość zwracana.

```
{ a: Int -> a * 3 }
```

Możesz nawet zapisać lambdę w zmiennej, jak pokazano na poniższym diagramie. Składnia jest podobna do deklarowania zmiennej podstawowego typu danych, takiego jak Int. Zwróć uwagę na nazwę zmiennej (żółte pole), typ zmiennej (niebieskie pole) i wartość zmiennej (zielone pole). Zmienna tripleprzechowuje funkcję. Jego typ to typ funkcji (Int) -> Int, a wartość to wyrażenie lambda { a: Int -> a * 3}.

1. Wypróbuj ten kod na placu zabaw. Zdefiniuj i wywołaj triplefunkcję, przekazując jej numer taki jak 5.

```
val number: Int = 5

val triple: (Int) -> Int = { a: Int -> a * 3 }
```

Function Type　　　　Lambda

```
fun main() {
   val triple: (Int) -> Int = { a: Int -> a * 3 }
   println(triple(5))
}
```

2. Wynikowy wynik powinien być:

```
15
```

Uwaga: Często zdarza się, że lambda ma jeden parametr, więc Kotlin oferuje skrót. Kotlin domyślnie używa specjalnego identyfikatora itdla parametru lambdy z jednym parametrem.

3. W nawiasach klamrowych można pominąć jawne deklarowanie parametru (a: Int), pominąć strzałkę funkcji (->) i po prostu mieć treść funkcji. Zaktualizuj triplefunkcję zadeklarowaną w swojej mainfunkcji i uruchom kod.

```
val triple: (Int) -> Int = { it * 3 }
```

4. Wynik powinien być taki sam, ale teraz twoja lambda jest napisana bardziej zwięźle! Więcej przykładów lambd znajdziesz w tym źródle .

```
15
```

Funkcje wyższego rzędu

Teraz, gdy zaczynasz dostrzegać elastyczność sposobu manipulowania funkcjami w Kotlinie, porozmawiajmy o innym naprawdę potężnym pomyśle, funkcji *wyższego rzędu* . Oznacza to po prostu przekazanie funkcji (w tym przypadku lambda) do innej funkcji lub zwrócenie funkcji z innej funkcji.

Okazuje się, że wszystkie funkcje map, filter, i forEachsą przykładami funkcji wyższego rzędu, ponieważ wszystkie przyjęły funkcję jako parametr. (W lambdzie przekazanej do tej filterfunkcji wyższego rzędu można pominąć pojedynczy parametr i symbol strzałki, a także użyć itparametru).

```
peopleAges.filter { it.key.length < 4 }
```

Oto przykład nowej funkcji wyższego rzędu: sortedWith().

Jeśli chcesz posortować listę ciągów, możesz użyć wbudowanej sorted()metody dla kolekcji. Jeśli jednak chcesz posortować listę według długości ciągów, musisz napisać kod, aby uzyskać długość dwóch ciągów i porównać je. Kotlin pozwala to zrobić, przekazując sortedWith()metodzie lambdę.

Uwaga: Aby porównać dwa obiekty do sortowania, konwencja polega na zwróceniu wartości mniejszej niż 0, jeśli pierwszy obiekt jest mniejszy niż drugi, 0, jeśli są równe, i wartości większej niż 0, jeśli pierwszy obiekt jest większy od drugiego .

1. Na placu zabaw utwórz listę nazw i wydrukuj ją posortowaną według nazwy za pomocą tego kodu:

```
fun main() {
    val peopleNames = listOf("Fred", "Ann", "Barbara", "Joe")
    println(peopleNames.sorted())
}
```

2. Teraz wydrukuj listę posortowaną według długości nazw, przekazując do sortedWith()funkcji lambdę. Lambda powinna przyjmować dwa parametry tego samego typu i zwracać an Int. Dodaj ten wiersz kodu po println()instrukcji w main()funkcji.

```
println(peopleNames.sortedWith { str1: String, str2: String -> str1.length - str2.length })
```

3. Uruchom swój program i spójrz na wyniki.

```
[Ann, Barbara, Fred, Joe]
[Ann, Joe, Fred, Barbara]
```

Przekazywana lambda sortedWith()ma dwa parametry: str1a Stringi str2a String. Następnie zobaczysz strzałkę funkcji, po której następuje treść funkcji.

```
{ str1: String, str2: String -> str1.length - str2.length }
```

Pamiętaj, że ostatnie wyrażenie w lambdzie to wartość zwracana. W tym przypadku zwraca różnicę między długością pierwszego ciągu a długością drugiego ciągu, czyli Int. To pasuje do tego, co jest potrzebne do sortowania: jeśli str1jest krótszy niż str2, zwróci wartość mniejszą niż 0. Jeśli str1i str2mają tę samą długość, zwróci 0. Jeśli str1jest dłuższy niż str2, zwróci wartość większą niż 0. seria porównań między dwoma Stringsnaraz, sortedWith()funkcja wypisuje listę, w której nazwy będą uporządkowane według rosnącej długości.

OnClickListener i OnKeyListener w systemie Android

Łącząc to z tym, czego nauczyłeś się do tej pory w systemie Android, używałeś lambd we wcześniejszych ćwiczeniach z programowania, na przykład podczas ustawiania odbiornika kliknięć dla przycisku w aplikacji Kalkulator wskazówek:

```
calculateButton.setOnClickListener{ calculateTip() }
```

Używanie lambda do ustawienia odbiornika kliknięć jest wygodnym skrótem. Długa forma pisania powyższego kodu jest pokazana poniżej i porównana z wersją skróconą. Nie musisz rozumieć wszystkich szczegółów wersji długiej, ale zwróć uwagę na pewne wzorce między tymi dwiema wersjami.

LONG FORM

```
calculateButton.setOnClickListener(object: View.OnClickListener {
    override fun onClick(view: View?) {
        calculateTip()
    }
})
```

SHORT FORM

```
calculateButton.setOnClickListener { view -> calculateTip() }
```

Obserwuj, jak lambda ma ten sam typ funkcji, co onClick()metoda in OnClickListener(przyjmuje jeden Viewargument i zwraca Unit, co oznacza brak zwracanej wartości).

Skrócona wersja kodu jest możliwa dzięki czemuś, co nazywa się konwersją SAM (Single-Abstract-Method) w Kotlinie. Kotlin konwertuje lambdę na OnClickListenerobiekt, który implementuje pojedynczą metodę abstrakcyjną onClick(). Musisz tylko upewnić się, że typ funkcji lambda pasuje do typu funkcji funkcji abstrakcyjnej.

Ponieważ viewparametr nigdy nie jest używany w lambdzie, można go pominąć. Następnie mamy tylko ciało funkcji w lambdzie.

```
calculateButton.setOnClickListener { calculateTip() }
```

Te koncepcje są trudne, więc bądź cierpliwy, ponieważ zagłębienie się w te koncepcje zajmie trochę czasu i doświadczenia. Spójrzmy na inny przykład. Przypomnij sobie, kiedy ustawisz detektor klawiszy w polu tekstowym „Koszt usługi" w kalkulatorze napiwków, aby klawiatura ekranowa mogła być ukryta po naciśnięciu klawisza Enter.

```
costOfServiceEditText.setOnKeyListener { view, keyCode, event -> handleKeyEvent(view, keyCode) }
```

Kiedy patrzysz w górę OnKeyListener, metoda abstrakcyjna ma następujące parametry onKey(View v, int keyCode, KeyEvent event)i zwraca Boolean. Ze względu na konwersje SAM w Kotlinie, możesz przekazać lambdę do setOnKeyListener(). Tylko upewnij się, że lambda ma typ funkcji (View, Int, KeyEvent) -> Boolean.

Oto diagram wyrażenia lambda użytego powyżej. Parametry to widok, kod klucza i zdarzenie. Treść funkcji składa się z handleKeyEvent(view, keyCode)której używa przekazanych parametrów i zwraca a Boolean.

```
{ view, keyCode, event -> handleKeyEvent(view, keyCode) }
```

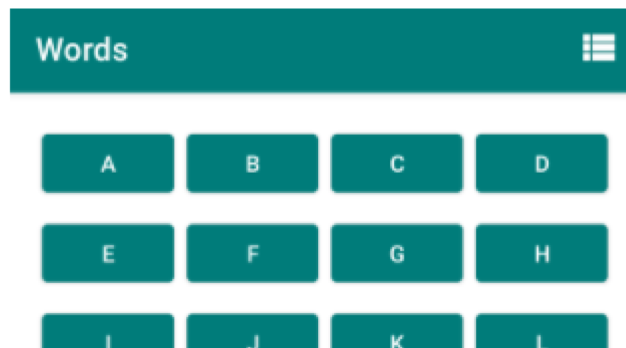
Uwaga: Jeśli nie używasz parametru lambda w treści funkcji, możesz go nazwać, _aby kod był bardziej czytelny i mniej zagracony. Ten kod ma takie samo zachowanie.

```
costOfServiceEditText.setOnKeyListener { view, keyCode, _ -> handleKeyEvent(view, keyCode) }
```

5. Twórz listy słów

Teraz weźmy wszystko, czego nauczyliście się o kolekcjach, lambdach i funkcjach wyższego rzędu, i zastosujmy to w realistycznym przypadku użycia.

Załóżmy, że chcesz stworzyć aplikację na Androida, aby grać w grę słowną lub uczyć się słówek. Aplikacja może wyglądać mniej więcej tak, z przyciskiem dla każdej litery alfabetu:



Kliknięcie na literę **A** spowoduje wyświetlenie krótkiej listy słów zaczynających się na literę A i tak dalej.

Będziesz potrzebować zbioru słów, ale jakiego rodzaju? Jeśli aplikacja ma zawierać słowa zaczynające się od każdej litery alfabetu, będziesz potrzebować sposobu na znalezienie lub uporządkowanie wszystkich słów zaczynających się na daną literę. Aby było to trudniejsze, wybierz inne słowa ze swojej kolekcji za każdym razem, gdy użytkownik uruchomi aplikację.

Najpierw zacznij od listy słów. W przypadku prawdziwej aplikacji potrzebujesz dłuższej listy słów i zawieraj słowa, które zaczynają się od wszystkich liter alfabetu, ale na razie wystarczy krótka lista.

1. Zastąp kod w placu zabaw Kotlin tym kodem:

```
fun main() {
    val words = listOf("about", "acute", "awesome", "balloon", "best", "brief", "class", "coffee", "creative")
}
```

2. Aby uzyskać zbiór słów zaczynających się na literę B, możesz użyć `filter`wyrażenia lambda. Dodaj te wiersze:

```
val filteredWords = words.filter { it.startsWith("b", ignoreCase = true) }
println(filteredWords)
```

Funkcja [startsWith()](#)zwraca prawdę, jeśli ciąg zaczyna się od określonego ciągu. Możesz również powiedzieć mu, aby ignorował wielkość liter, więc „b” będzie pasować do „b” lub „B”.

3. Uruchom swój program i spójrz na wynik:

```
[balloon, best, brief]
```

4. Pamiętaj, że chcesz, aby słowa były losowo używane w Twojej aplikacji. Dzięki kolekcji Kotlin możesz użyć [shuffled()](#)funkcji do wykonania kopii kolekcji z losowo potasowanymi elementami. Zmień również filtrowane słowa na tasowane:

```
val filteredWords = words.filter { it.startsWith("b", ignoreCase = true) }
    .shuffled()
```

5. Uruchom swój program i spójrz na nowe wyniki:

   [brief, balloon, best]

   Ponieważ słowa są losowo przetasowane, możesz zobaczyć je w innej kolejności.

6. Nie chcesz wszystkich słów (zwłaszcza jeśli twoja prawdziwa lista słów jest długa), tylko kilka. Możesz użyć take()funkcji, aby uzyskać pierwsze pozycje w kolekcji. Spraw, aby filtrowane słowa zawierały tylko pierwsze dwa potasowane słowa:

```
val filteredWords = words.filter { it.startsWith("b", ignoreCase = true) }
    .shuffled()
    .take(2)
```

7. Uruchom swój program i spójrz na nowe wyniki:

   [brief, balloon]

   Ponownie, z powodu losowego tasowania, za każdym razem możesz zobaczyć inne słowa.

8. Wreszcie, dla aplikacji chcesz posortować losową listę słów dla każdej litery. Tak jak poprzednio, możesz użyć sorted()funkcji do zwrócenia kopii kolekcji z posortowanymi elementami:

```
val filteredWords = words.filter { it.startsWith("b", ignoreCase = true) }
    .shuffled()
    .take(2)
    .sorted()
```

9. Uruchom swój program i spójrz na nowe wyniki:

   [balloon, brief]

   Cały powyższy kod:

```
fun main() {
    val words = listOf("about", "acute", "awesome", "balloon", "best", "brief", "class", "coffee", "creative")
    val filteredWords = words.filter { it.startsWith("b", ignoreCase = true) }
        .shuffled()
        .take(2)
        .sorted()
    println(filteredWords)
}
```

10. Spróbuj zmienić kod, aby utworzyć listę jednego losowego słowa, które zaczyna się na literę c. Co musisz zmienić w powyższym kodzie?

```
val filteredWords = words.filter { it.startsWith("c", ignoreCase = true) }
    .shuffled()
    .take(1)
```

   W rzeczywistej aplikacji musisz zastosować filtr dla każdej litery alfabetu, ale teraz wiesz, jak wygenerować listę słów dla każdej litery!

Kolekcje są potężne i elastyczne. Mogą zrobić wiele i może być więcej niż jeden sposób, aby coś zrobić. Gdy dowiesz się więcej o programowaniu, dowiesz się, jak ustalić, który typ kolekcji jest odpowiedni dla danego problemu i jak najlepiej go przetworzyć.

Lambdy i funkcje wyższego rzędu sprawiają, że praca z kolekcjami jest łatwiejsza i bardziej zwięzła. Te pomysły są bardzo przydatne, więc będziesz ich używać wielokrotnie.

6. Podsumowanie

- Kolekcja to grupa powiązanych elementów
- Kolekcje mogą być zmienne lub niezmienne
- Kolekcje można zamawiać lub nie zamawiać
- Kolekcje mogą wymagać unikalnych przedmiotów lub zezwalać na duplikaty
- Kotlin obsługuje różne rodzaje kolekcji, w tym listy, zestawy i mapy
- Kotlin udostępnia wiele funkcji do przetwarzania i przekształcania kolekcji, w tym `forEach`, `map`, `filter`, `sorted`i nie tylko.
- Lambda to funkcja bez nazwy, którą można od razu przekazać jako wyrażenie. Przykładem może być { a: Int -> a * 3 }.
- Funkcja wyższego rzędu oznacza przekazanie funkcji do innej funkcji lub zwrócenie funkcji z innej funkcji.

7. Dowiedz się więcej

- Słownictwo dla Androida Podstawy w Kotlin
- Kolekcje Kotlina
- `List`klasa
- `Set`klasa
- `Map`klasa
- `Collection`przekształcenia
- Funkcje wyższego rzędu i lambdy
- Rodzaje funkcji
- it: niejawna nazwa dla pojedynczego parametru
- Funkcje lambda
- Funkcje wyższego rzędu

Działania i zamiary

1. Wstęp

Do tej pory aplikacje, nad którymi pracowałeś, miały tylko jedną aktywność. W rzeczywistości wiele aplikacji na Androida wymaga wielu czynności z nawigacją między nimi.

W tym ćwiczeniu z programowania zbudujesz aplikację słownika, aby korzystała z wielu działań, wykorzystywała intencje do nawigowania między nimi i przekazywała dane do innych aplikacji.

Warunki wstępne

Powinieneś być w stanie:

- Poruszaj się po projekcie w Android Studio.
- Pracuj z zasobami XML i dodawaj je w Android Studio.
- Zastąp i zaimplementuj metody w istniejącej klasie.
- Twórz instancje klas Kotlin, właściwości klas dostępu i metody wywoływania.
- Zapoznaj się z dokumentacją na stronie developer.android.com , aby dowiedzieć się więcej o konkretnych klasach.

Czego się nauczysz

Jak:

- Użyj wyraźnej intencji, aby przejść do określonej aktywności.
- Użyj niejawnej intencji, aby przejść do treści w innej aplikacji.
- Dodaj opcje menu, aby dodać przyciski do paska aplikacji.

Co zbudujesz

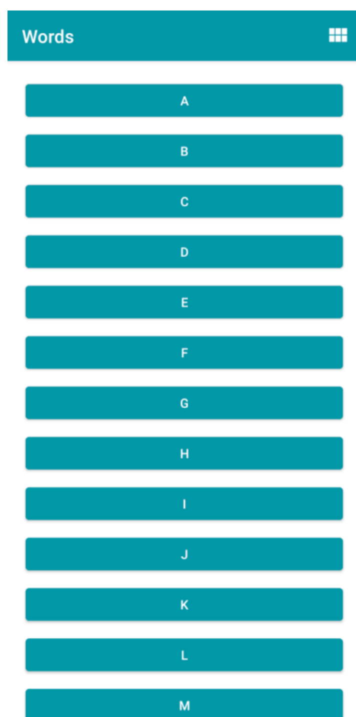
- Zmodyfikuj aplikację słownika, aby zaimplementować nawigację między ekranami za pomocą intencji i dodając menu opcji.

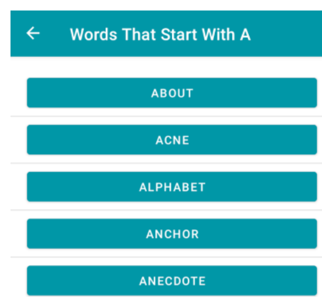
Czego potrzebujesz

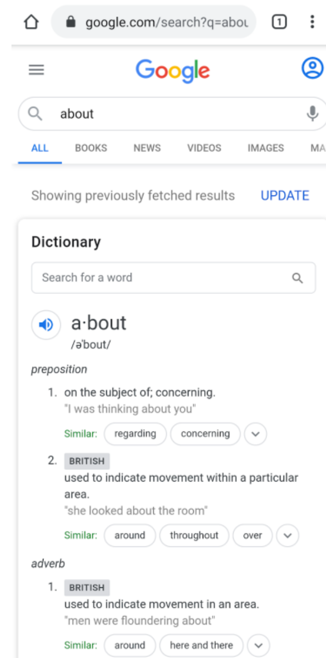
- Komputer z zainstalowanym Android Studio.

2. Kod startowy

W kilku kolejnych krokach będziesz pracować w aplikacji Words. Aplikacja Words to prosta aplikacja słownikowa z listą liter, słów dla każdej litery i możliwością wyszukiwania definicji każdego słowa w przeglądarce.







Dużo się dzieje, ale nie martw się — nie musisz tworzyć całej aplikacji tylko po to, by poznać intencje. Zamiast tego otrzymałeś niekompletną wersję projektu lub projekt startowy.

Chociaż wszystkie ekrany są zaimplementowane, nie można jeszcze przechodzić z jednego ekranu na drugi. Twoim zadaniem jest wykorzystanie intencji, aby cały projekt działał, bez konieczności budowania wszystkiego od zera.

Pobierz kod startowy do tego ćwiczenia z programowania

To ćwiczenie z programowania zapewnia kod startowy, który można rozszerzyć o funkcje nauczane w tym ćwiczeniu z programowania. Kod startowy może zawierać kod, który jest Ci znany z poprzednich ćwiczeń z programowania. Może również zawierać kod, który jest Tobie nieznany i o którym dowiesz się podczas późniejszych ćwiczeń z programowania.

Po pobraniu kodu startowego z usługi GitHub zwróć uwagę, że nazwa folderu to `android-basics-kotlin-words-app-starter`. Wybierz ten folder podczas otwierania projektu w Android Studio.

URL kodu startowego: https://github.com/google-developer-training/android-basics-kotlin-words-app/tree/starter

Nazwa oddziału w GitHub:

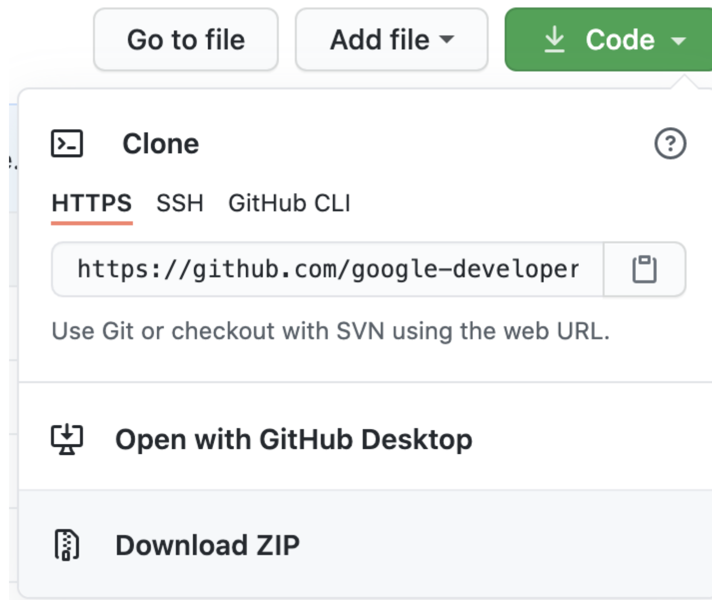
rozrusznik

Jeśli znasz polecenia git, zauważ, że kod startowy znajduje się w gałęzi o nazwie „starter”. Po sklonowaniu repozytorium sprawdź kod z `origin/starter`oddziału. Jeśli nie używałeś wcześniej poleceń git, wykonaj poniższe kroki, aby pobrać kod z GitHub.

Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

Pobierz kod

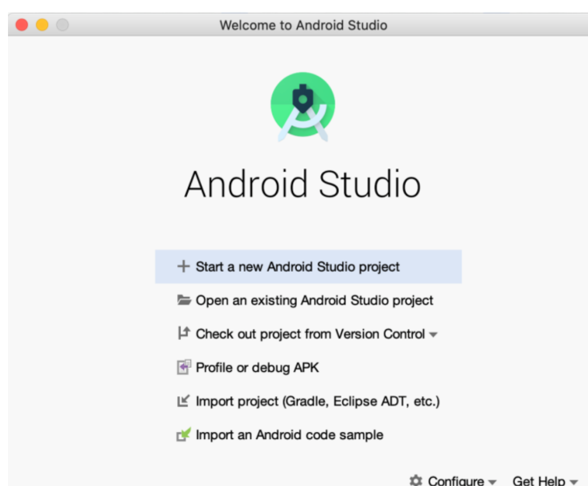
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



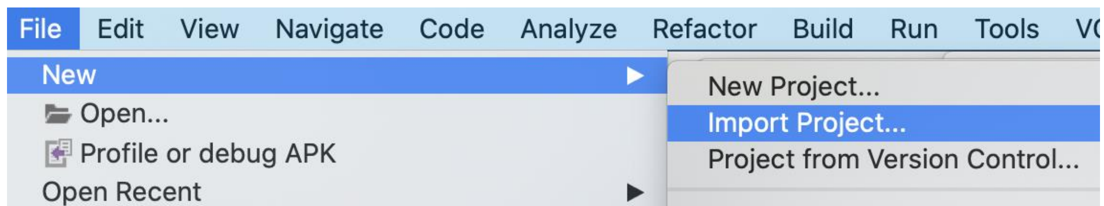
3. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
4. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
5. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

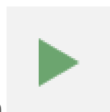
Otwórz projekt w Android Studio

1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz istniejący projekt Android Studio.**



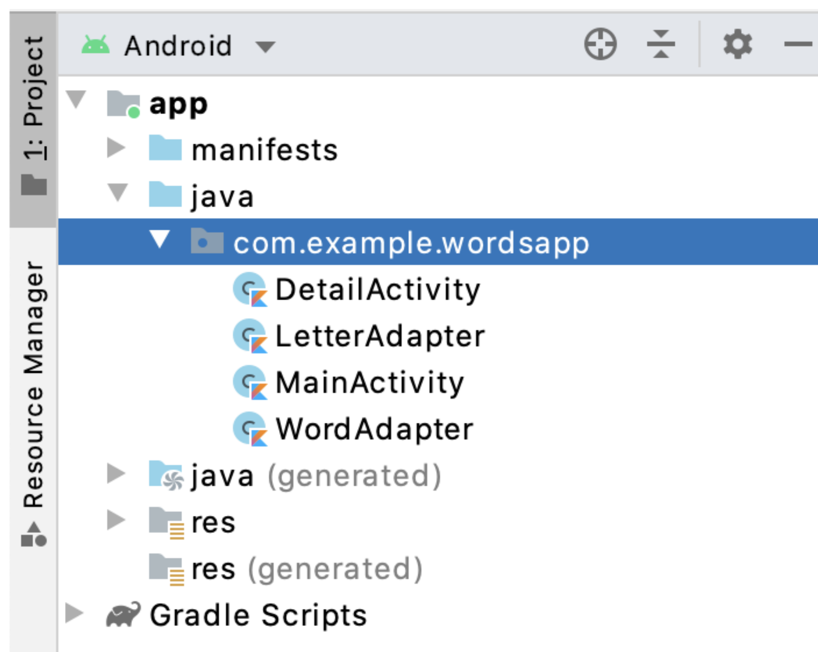
Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Nowy** > **Importuj projekt** .



3. W oknie dialogowym **Importuj projekt** przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.
6. Kliknij przycisk **Uruchom**  , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.
7. Przeglądaj pliki projektu w oknie narzędzia **Projekt** , aby zobaczyć, jak skonfigurowana jest aplikacja.

3. Przegląd aplikacji Words

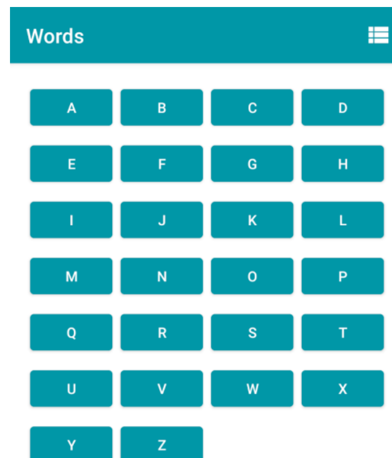
Zanim przejdziesz dalej, poświęć chwilę na zapoznanie się z projektem. Wszystkie koncepcje powinny być znane z poprzedniej jednostki. Obecnie aplikacja składa się z dwóch czynności, z których każda ma widok recyklera i adapter.



Będziesz pracować w szczególności z następującymi plikami:

1. LetterAdapterjest używany przez RecyclerViewin MainActivity. Każda litera to przycisk z ikoną onClickListener, która obecnie jest pusta. Tutaj będziesz obsługiwał naciśnięcia przycisków, aby przejść do DetailActivity.

2. WordAdapterjest używany przez RecyclerViewin DetailActivitydo wyświetlania listy słów. Chociaż nie możesz jeszcze przejść do tego ekranu, po prostu pamiętaj, że każde słowo ma również odpowiedni przycisk ze znakiem onClickListener. Tutaj dodasz kod, który przechodzi do przeglądarki, aby pokazać definicję słowa.
3. MainActivitybędzie również potrzebować kilku zmian. W tym miejscu zaimplementujesz menu opcji, aby wyświetlić przycisk, który pozwala użytkownikom przełączać się między układami listy i siatki.



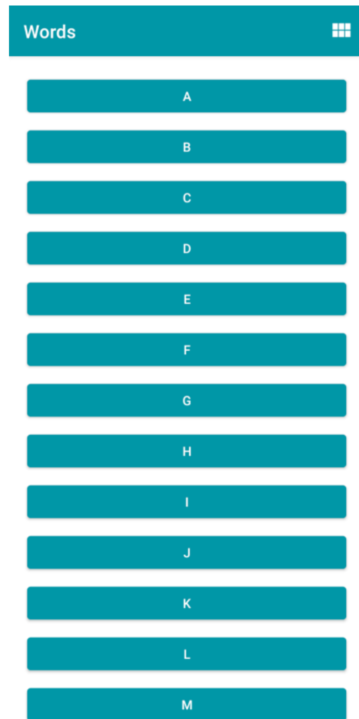
Gdy poczujesz się już komfortowo z projektem, przejdź do następnej sekcji, w której poznasz intencje.

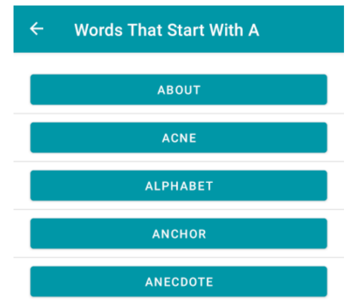
4. Wprowadzenie do intencji

Teraz, po skonfigurowaniu początkowego projektu, omówmy intencje i sposób ich wykorzystania w aplikacji.

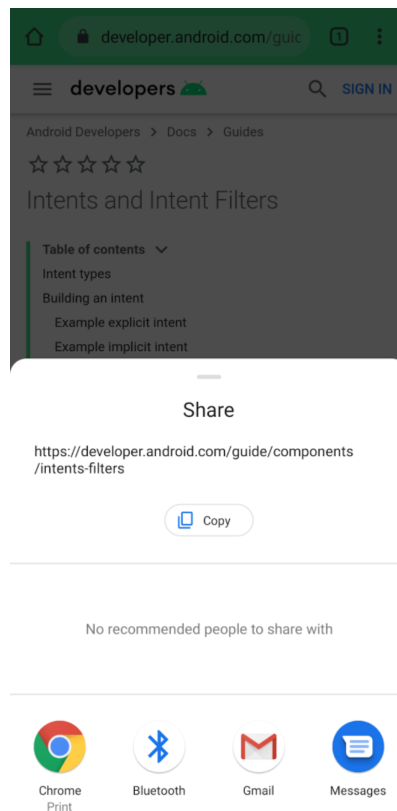
Intencja to obiekt reprezentujący jakąś akcję do wykonania. Najczęstszym, ale z pewnością nie jedynym, zastosowaniem do intencji jest uruchomienie działalności. Istnieją dwa rodzaje intencji — **niejawna** i **jawna** . Wyraźna **intencja** jest bardzo konkretna, gdzie dokładnie znasz działanie, które ma zostać uruchomione, często ekran we własnej aplikacji.

Niejawna **intencja** jest nieco bardziej abstrakcyjna, gdy mówisz systemowi o typie działania, takim jak otwarcie łącza, napisanie wiadomości e-mail lub wykonanie połączenia telefonicznego, a system jest odpowiedzialny za ustalenie, jak spełnić żądanie. Prawdopodobnie widziałeś oba rodzaje intencji w działaniu, nawet o tym nie wiedząc. Generalnie, pokazując aktywność we własnej aplikacji, używasz wyraźnej intencji.

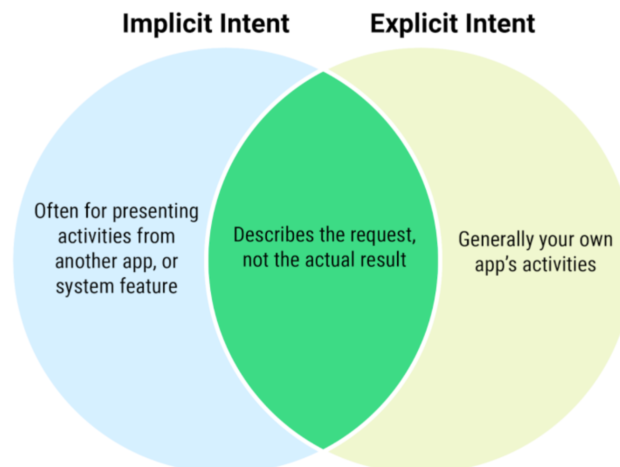




Jednak w przypadku działań, które niekoniecznie dotyczą bieżącej aplikacji — powiedzmy, że znalazłeś interesującą stronę z dokumentacją Androida i chcesz ją udostępnić znajomym — użyj **niejawnej intencji** . Możesz zobaczyć takie menu z pytaniem, której aplikacji użyć do udostępnienia strony.



Używasz wyraźnej intencji do działań lub prezentowania ekranów we własnej aplikacji i jesteś odpowiedzialny za cały proces. Często używasz niejawnych intencji do wykonywania działań związanych z innymi aplikacjami i polegasz na systemie, aby określić wynik końcowy. W aplikacji Words będziesz używać obu typów intencji.



5. Skonfiguruj wyraźną intencję

Czas na realizację pierwszego zamiaru. Na pierwszym ekranie, gdy użytkownik dotknie litery, powinien zostać przeniesiony na drugi ekran z listą słów. Jest DetailActivityjuż zaimplementowany, więc wystarczy uruchomić go z zamiarem. Ponieważ Twoja aplikacja dokładnie wie, jaką aktywność należy uruchomić, używasz wyraźnej intencji.

Tworzenie i używanie intencji to tylko kilka kroków:

1. Otwórz LetterAdapter.kti przewiń w dół do onBindViewHolder(). Poniżej linii, aby ustawić tekst przycisku, ustaw onClickListenerfor holder.button.

```
holder.button.setOnClickListener {

}
```

2. Następnie uzyskaj odwołanie do context.

```
val context = holder.view.context
```

3. Utwórz Intent, przekazując w kontekście, a także nazwę klasy działania docelowego.

```
val intent = Intent(context, DetailActivity::class.java)
```

Nazwa działania, które chcesz pokazać, jest określona za pomocą DetailActivity::class.java. DetailActivityZa kulisami powstaje rzeczywisty obiekt.

4. Wywołaj putExtrametodę, przekazując „literę" jako pierwszy argument i tekst przycisku jako drugi argument.

```
intent.putExtra("letter", holder.button.text.toString())
```

Co to jest dodatek? Pamiętaj, że intencja to po prostu zestaw instrukcji — nie ma jeszcze wystąpienia działania docelowego. Zamiast tego, dodatek to fragment danych, taki jak liczba lub ciąg, któremu nada się nazwę, którą można później pobrać. Jest to podobne do przekazywania argumentu podczas wywoływania funkcji. Ponieważ a DetailActivitymoże być pokazane dla dowolnej litery, musisz powiedzieć, którą literę przedstawić.

Jak myślisz, dlaczego trzeba zadzwonić toString()? Tekst przycisku jest już ciągiem, prawda?

Raczej. W rzeczywistości jest typu CharSequence, co nazywa się *interfejsem* . Na razie nie musisz nic wiedzieć o interfejsach Kotlina, poza tym, że jest to sposób na upewnienie się, że typ, taki jak String, implementuje określone funkcje i właściwości. Możesz myśleć o a CharSequencejako o bardziej ogólnej reprezentacji klasy podobnej do łańcucha. Własnością przycisku textmoże być ciąg znaków lub dowolny obiekt, który jest również CharSequence. Metoda putExtra()akceptuje jednak a String, a nie byle CharSequencejaki , stąd konieczność wywołania toString().

5. Wywołaj startActivity()metodę w obiekcie kontekstu, przekazując intent.

```
context.startActivity(intent)
```

Teraz uruchom aplikację i spróbuj dotknąć litery. Wyświetlany jest ekran szczegółów! Ale bez względu na to, którą literę kliknie użytkownik, ekran szczegółów zawsze pokazuje słowa odpowiadające literze A. Nadal masz trochę pracy do wykonania w czynnościach dotyczących szczegółów, tak aby wyświetlały słowa dla dowolnej litery podanej jako intentdodatkowa.

6. Skonfiguruj aktywność szczegółów

Właśnie stworzyłeś swoją pierwszą wyraźną intencję! Teraz na ekranie szczegółów.

W onCreatemetodzie DetailActivity, po wywołaniu setContentView, zastąp zakodowaną na stałe literę kodem, aby pobrać letterIdprzekazaną z intent.

```
val letterId = intent?.extras?.getString("letter").toString()
```

Dużo się tutaj dzieje, więc spójrzmy na każdą część:

Po pierwsze, skąd intentpochodzi nieruchomość? Nie jest własnością DetailActivity, ale raczej własnością jakiejkolwiek działalności. Zachowuje odniesienie do intencji użytej do uruchomienia działania.

Własność extras jest typu Bundlei, jak można się domyślić, zapewnia dostęp do wszystkich dodatków przekazanych do intencji.

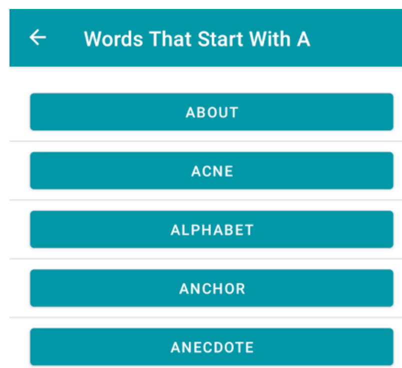
Obie te właściwości są oznaczone znakiem zapytania. Dlaczego to? Powodem jest to, że właściwości intenti extrasmają wartość null, co oznacza, że mogą mieć wartość lub nie. Czasami możesz chcieć, aby zmienna była null. Właściwość intentmoże w rzeczywistości nie być Intent(jeśli działanie nie zostało uruchomione z zamiaru), a właściwość extras może w rzeczywistości nie być wartością Bundle, ale raczej wartością o nazwie null. W Kotlinie nulloznacza brak wartości. Obiekt może istnieć lub może być null. Jeśli Twoja aplikacja spróbuje uzyskać dostęp do właściwości lub wywołać funkcję w nullobiekcie, aplikacja ulegnie awarii. Aby bezpiecznie uzyskać dostęp do tej wartości, umieść znak ?po nazwie. Jeśli intentjest null, Twoja aplikacja nie będzie nawet próbowała uzyskać dostępu do właściwości dodatków, a jeśliextrasma wartość null, Twój kod nie będzie nawet próbował wywołać getString().

Skąd wiesz, które właściwości wymagają znaku zapytania, aby zapewnić zerowe bezpieczeństwo? Możesz stwierdzić, czy po nazwie typu występuje znak zapytania lub wykrzyknik.

```
intent?.extra
    extras (from get…   Bundle?
```

Ostatnią rzeczą, na którą należy zwrócić uwagę, jest to, że faktycznie pobierana jest litera getString, która zwraca a String?, więc toString()jest wywoływana, aby upewnić się, że jest to a String, a nie null.

Teraz, gdy uruchomisz aplikację i przejdziesz do ekranu szczegółów, powinieneś zobaczyć listę słów dla każdej litery.



Sprzątanie

Zarówno kod, aby wykonać intencję, jak i pobrać wybraną literę na sztywno z nazwą extra„litery". Chociaż działa to w tym małym przykładzie, nie jest to najlepsze podejście w przypadku dużych aplikacji, w których masz o wiele więcej intencjonalnych dodatków do śledzenia.

Chociaż możesz po prostu utworzyć stałą o nazwie „litera", może to stać się niewygodne, gdy dodasz do swojej aplikacji więcej celowych dodatków. A w jakiej klasie umieściłbyś tę stałą? Pamiętaj, że ciąg jest używany w obu DetailActivityi MainActivity. Potrzebujesz sposobu na zdefiniowanie stałej, która może być używana w wielu klasach, przy jednoczesnym zachowaniu porządku w kodzie.

Na szczęście istnieje przydatna funkcja Kotlin, której można użyć do oddzielenia stałych i uczynienia ich użytecznymi bez określonej instancji klasy zwanej *obiektami towarzyszącymi* . Obiekt towarzyszący jest podobny do innych obiektów, takich jak instancje klasy. Jednak tylko jedna instancja obiektu towarzyszącego będzie istnieć przez czas trwania programu, dlatego jest to czasami nazywane *wzorcem singleton* . Chociaż istnieje wiele przypadków użycia *pojedynczych* elementów poza zakresem tego ćwiczenia z programowania, na razie użyjesz obiektu towarzyszącego jako sposobu na uporządkowanie stałych i udostępnienie ich poza DetailActivity. Zaczniesz od użycia obiektu towarzyszącego, aby zrefaktoryzować kod dodatkowego „litery".

1. W DetailActivity, tuż powyżej onCreate, dodaj następujące informacje:

```
companion object {

}
```

   Zauważ, że jest to podobne do definiowania klasy, z wyjątkiem tego, że używasz objectsłowa kluczowego. Istnieje również słowo kluczowe companion, co oznacza, że jest powiązane z DetailActivityklasą i nie musimy nadawać mu osobnej nazwy typu.

2. W nawiasach klamrowych dodaj właściwość stałej litery.

```
const val LETTER = "letter"
```

3. Aby użyć nowej stałej, zaktualizuj swoje zakodowane wywołanie literowe w onCreate()następujący sposób:

```
val letterId = intent?.extras?.getString(LETTER).toString()
```

   Ponownie zauważ, że jak zwykle odwołujesz się do niej za pomocą notacji kropkowej, ale stała należy do DetailActivity.

4. Przełącz się na LetterAdapteri zmodyfikuj wywołanie do, putExtraaby używało nowej stałej.

```
intent.putExtra(DetailActivity.LETTER, holder.button.text.toString())
```

Wszystko gotowe! Refaktoryzacja sprawia, że kod jest łatwiejszy do odczytania i łatwiejszy w utrzymaniu. Jeśli ta lub jakakolwiek inna stała, którą dodasz, kiedykolwiek będzie musiała się zmienić, musisz to zrobić tylko w jednym miejscu.

Aby dowiedzieć się więcej o obiektach towarzyszących, zapoznaj się z dokumentacją Kotlina dotyczącą [wyrażeń i deklaracji obiektów](#) .

7. Stwórz ukrytą intencję

W większości przypadków przedstawiasz konkretne działania z własnej aplikacji. Są jednak sytuacje, w których możesz nie wiedzieć, jaką aktywność lub jaką aplikację chcesz uruchomić. Na naszym ekranie szczegółowym każde słowo jest przyciskiem, który pokaże definicję słowa użytkownika.

W naszym przykładzie użyjesz funkcji słownika udostępnianej przez wyszukiwarkę Google. Jednak zamiast dodawać nową aktywność do aplikacji, uruchomisz przeglądarkę urządzenia, aby wyświetlić stronę wyszukiwania.

Może więc potrzebujesz zamiaru załadowania strony w Chrome, domyślnej przeglądarce na Androidzie?

Nie do końca.

Możliwe, że niektórzy użytkownicy wolą przeglądarkę innej firmy. Lub ich telefon jest wyposażony w przeglądarkę preinstalowaną przez producenta. Być może mają zainstalowaną wyszukiwarkę Google, a nawet słownik innej firmy.

Nie możesz wiedzieć na pewno, jakie aplikacje zainstalował użytkownik. Nie możesz też zakładać, jak mogą chcieć wyszukać słowo. Jest to doskonały przykład tego, kiedy należy użyć niejawnej intencji. Twoja aplikacja dostarcza systemowi informacji o tym, jaka akcja powinna być, a system zastanawia się, co z nią zrobić, prosząc użytkownika o dodatkowe informacje w razie potrzeby.

Wykonaj następujące czynności, aby utworzyć niejawną intencję:

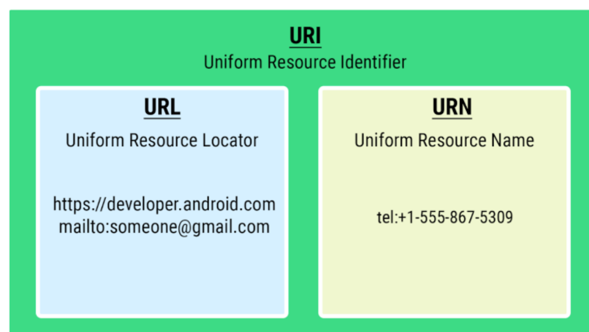
1. W przypadku tej aplikacji wyszukasz słowo w Google. Pierwszym wynikiem wyszukiwania będzie słownikowa definicja słowa. Ponieważ ten sam podstawowy adres URL jest używany do każdego wyszukiwania, dobrym pomysłem jest zdefiniowanie go jako własnej stałej. W `DetailActivity`programie zmodyfikuj obiekt towarzyszący, aby dodać nową stałą, `SEARCH_PREFIX`. To jest podstawowy adres URL wyszukiwarki Google.

```
companion object {
  const val LETTER = "letter"
  const val SEARCH_PREFIX = "https://www.google.com/search?q="
}
```

2. Następnie otwórz `WordAdapter`i w `onBindViewHolder()`metodzie wywołaj `setOnClickListener()`przycisk. Zacznij od utworzenia `URI`zapytania wyszukiwania. Podczas wywoływania `parse()`tworzenia a `URI`z a `String`, musisz użyć formatowania ciągu, aby słowo zostało dołączone do `SEARCH_PREFIX`.

```
holder.button.setOnClickListener {
   val queryUrl: Uri = Uri.parse("${DetailActivity.SEARCH_PREFIX}${item}")
}
```

Jeśli zastanawiasz się, czym jest *identyfikator URI* , nie jest to literówka, ale oznacza *Uniform Resource Identifier* . Być może już wiesz, że adres URL lub *Uniform Resource Locator* to ciąg znaków wskazujący stronę internetową. URI to bardziej ogólny termin określający format. Wszystkie adresy URL są identyfikatorami URI, ale nie wszystkie identyfikatory URI są adresami URL. Inne identyfikatory URI, na przykład adres dla numeru telefonu, zaczynają się od `tel:`, ale jest to URN lub *Uniform Resource Name* , a nie adres URL. Typ danych używany do reprezentowania obu jest nazywany `URI`.



Zwróć uwagę, że nie ma tutaj odniesienia do jakiejkolwiek aktywności w Twojej własnej aplikacji. Po prostu podajesz `URI`, bez wskazania, w jaki sposób jest on ostatecznie używany.

2. Po zdefiniowaniu `queryUrl`zainicjuj nowy `intent`obiekt:

```
val intent = Intent(Intent.ACTION_VIEW, queryUrl)
```

Zamiast przekazywać kontekst i czynność, przekazujesz `Intent.ACTION_VIEW`wraz z `URI`.

`ACTION_VIEW`to ogólna intencja, która przyjmuje identyfikator URI, w twoim przypadku adres internetowy. Następnie system wie, jak przetworzyć tę intencję, otwierając identyfikator URI w przeglądarce internetowej użytkownika. Niektóre inne typy intencji obejmują:

- `CATEGORY_APP_MAPS`– uruchomienie aplikacji map
- `CATEGORY_APP_EMAIL`– uruchomienie aplikacji e-mail
- `CATEGORY_APP_GALLERY`– uruchomienie aplikacji Galeria (zdjęcia)
- `ACTION_SET_ALARM`– ustawienie alarmu w tle
- `ACTION_DIAL`– inicjowanie rozmowy telefonicznej

Aby dowiedzieć się więcej, zapoznaj się z dokumentacją dotyczącą niektórych [często używanych intencji](#) .

3. Na koniec, nawet jeśli nie uruchamiasz żadnej konkretnej aktywności w swojej aplikacji, mówisz systemowi, aby uruchomił inną aplikację, wywołując `startActivity()`i przekazując `intent`.

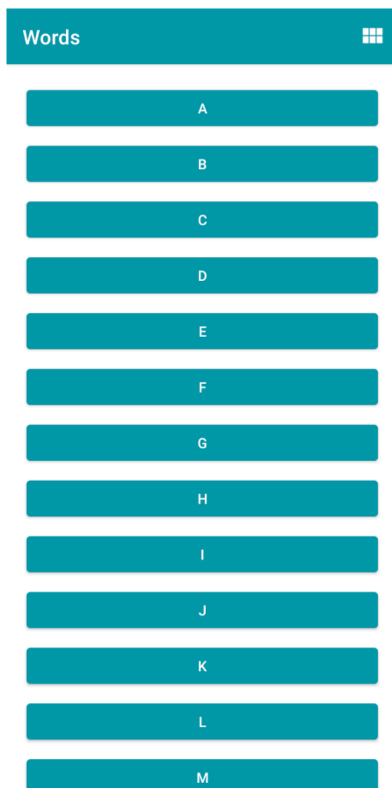
context.startActivity(intent)

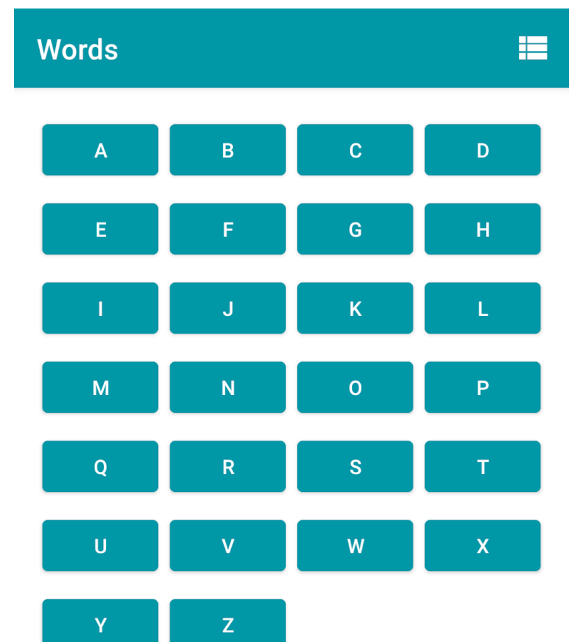
Teraz po uruchomieniu aplikacji przejdź do listy słów i dotknij jednego ze słów, urządzenie powinno przejść do adresu URL (lub wyświetlić listę opcji w zależności od zainstalowanych aplikacji).

Dokładne zachowanie będzie się różnić między użytkownikami, zapewniając bezproblemową obsługę dla wszystkich, bez komplikowania kodu.

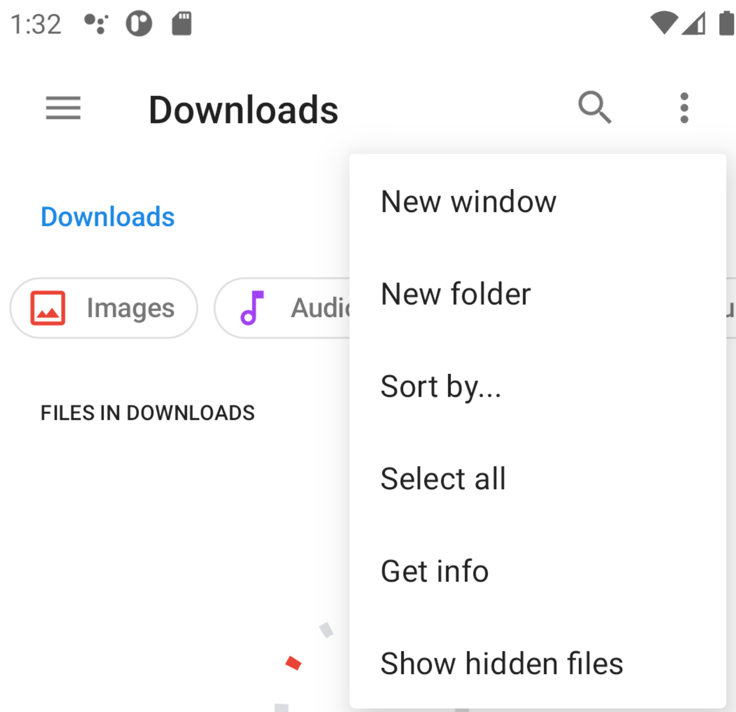
8. Skonfiguruj menu i ikony

Teraz, gdy masz już pełną nawigację po aplikacji, dodając wyraźne i niejawne intencje, nadszedł czas, aby dodać opcję menu, aby użytkownik mógł przełączać się między układami listy i siatki dla liter.



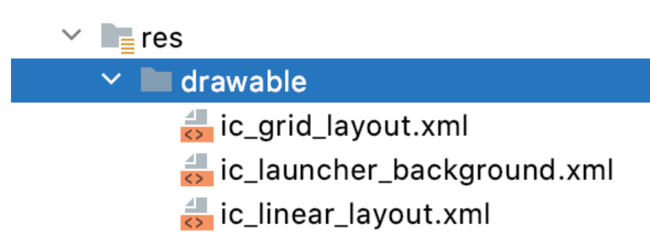


Do tej pory prawdopodobnie zauważyłeś, że wiele aplikacji ma ten pasek u góry ekranu. Nazywa się to paskiem aplikacji i oprócz wyświetlania nazwy aplikacji, pasek aplikacji można dostosować i udostępniać wiele przydatnych funkcji, takich jak skróty do przydatnych działań lub rozszerzone menu.

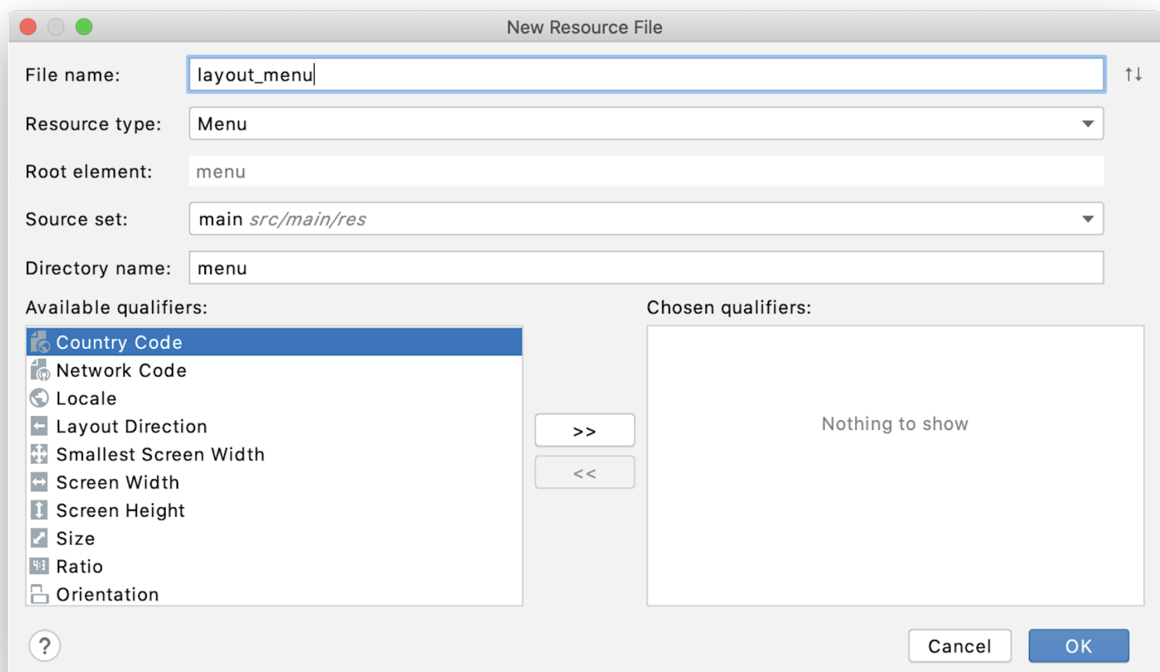


W przypadku tej aplikacji, chociaż nie dodamy pełnego menu, dowiesz się, jak dodać niestandardowy przycisk do paska aplikacji, aby użytkownik mógł zmienić układ.

1. Najpierw musisz zaimportować dwie ikony reprezentujące widoki siatki i listy. Dodaj zasoby wektorowe clipart o nazwie „moduł widoku” (nazwij go **ic_grid_layout**) i „lista widoków” (nazwij go **ic_linear_layout**). Jeśli potrzebujesz odświeżenia na temat dodawania ikon materiałów, zapoznaj się z instrukcjami na tej stronie .



2. Potrzebujesz również sposobu na poinformowanie systemu, jakie opcje są wyświetlane na pasku aplikacji i jakich ikon użyć. Aby to zrobić, dodaj nowy plik zasobów, klikając prawym przyciskiem myszy folder **res** i wybierając **Nowy > Plik zasobów systemu Android** . Ustaw **typ zasobu** na Menui nazwę **pliku** na layout_menu.



3. Kliknij **OK** .
4. Otwórz **res/Menu/layout_menu** . Zastąp zawartość `layout_menu.xml`następującą treścią:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto">
  <item android:id="@+id/action_switch_layout"
     android:title="@string/action_switch_layout"
     android:icon="@drawable/ic_linear_layout"
     app:showAsAction="always" />
</menu>
```

Struktura pliku menu jest dość prosta. Podobnie jak układ zaczyna się od menedżera układu do przechowywania poszczególnych widoków, plik XML menu zaczyna się od znacznika menu, który zawiera indywidualne opcje.

Twoje menu ma tylko jeden przycisk z kilkoma właściwościami:

- `id`: Podobnie jak widoki, opcja menu ma identyfikator, dzięki czemu można się do niej odwoływać w kodzie.
- `title`: Ten tekst w rzeczywistości nie będzie widoczny w Twoim przypadku, ale może być przydatny dla czytników ekranu do identyfikacji menu
- `icon`: Wartość domyślna to `ic_linear_layout`. Będzie to jednak włączane i wyłączane, aby wyświetlić ikonę siatki po wybraniu przycisku.
- `showAsAction`: To mówi systemowi, jak wyświetlić przycisk. Ponieważ jest ustawiony na zawsze, ten przycisk będzie zawsze widoczny na pasku aplikacji i nie stanie się częścią rozszerzonego menu.

Oczywiście samo ustawienie właściwości nie oznacza, że menu faktycznie nic nie zrobi.

Nadal będziesz musiał dodać trochę kodu MainActivity.kt, aby menu działało.

9. Przycisk Menu Wdrażania

Aby zobaczyć, jak działa przycisk menu, musisz zrobić kilka rzeczy w MainActivity.kt.

1. Po pierwsze, dobrym pomysłem jest utworzenie właściwości, aby śledzić, w jakim stanie układu znajduje się aplikacja. Ułatwi to przełączanie przycisku układu. Ustaw wartość domyślną na true, ponieważ menedżer układu liniowego będzie używany domyślnie.

```
private var isLinearLayoutManager = true
```

2. Gdy użytkownik przełączy przycisk, chcesz, aby lista elementów zmieniła się w siatkę elementów. Jeśli pamiętasz z nauki o widokach recyklera, istnieje wiele różnych menedżerów układu, z których jeden GridLayoutManagerpozwala na umieszczenie wielu elementów w jednym wierszu.

```
private fun chooseLayout() {
   if (isLinearLayoutManager) {
      recyclerView.layoutManager = LinearLayoutManager(this)
   } else {
      recyclerView.layoutManager = GridLayoutManager(this, 4)
   }
   recyclerView.adapter = LetterAdapter()
}
```

Tutaj używasz ifinstrukcji, aby przypisać menedżera układu. Oprócz ustawienia layoutManager, ten kod przypisuje również adapter. LetterAdapterjest używany zarówno w przypadku układów list, jak i siatki.

3. Podczas początkowej konfiguracji menu w xml nadałeś mu statyczną ikonę. Jednak po przełączeniu układu należy zaktualizować ikonę, aby odzwierciedlała jej nową funkcję — przełączenie z powrotem do układu listy. Tutaj po prostu ustaw ikony układu liniowego i siatki, w oparciu o układ, do którego przycisk przełączy się z powrotem przy następnym dotknięciu.

```
private fun setIcon(menuItem: MenuItem?) {
  if (menuItem == null)
     return

  // Set the drawable for the menu icon based on which LayoutManager is currently in use

  // An if-clause can be used on the right side of an assignment if all paths return a value.
  // The following code is equivalent to
  // if (isLinearLayoutManager)
  //     menu.icon = ContextCompat.getDrawable(this, R.drawable.ic_grid_layout)
  // else menu.icon = ContextCompat.getDrawable(this, R.drawable.ic_linear_layout)
```

```
  menuItem.icon =
     if (isLinearLayoutManager)
         ContextCompat.getDrawable(this, R.drawable.ic_grid_layout)
     else ContextCompat.getDrawable(this, R.drawable.ic_linear_layout)
}
```

Ikona jest ustawiana warunkowo na podstawie `isLinearLayoutManager`właściwości.

Aby Twoja aplikacja faktycznie korzystała z menu, musisz zastąpić dwie dodatkowe metody.

- `onCreateOptionsMenu`: gdzie napełniasz menu opcji i wykonujesz dodatkowe ustawienia
- `onOptionsItemSelected`: gdzie faktycznie będziesz dzwonić `chooseLayout()`po wybraniu przycisku.

1. Zastąp `onCreateOptionsMenu`w następujący sposób:

```
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
  menuInflater.inflate(R.menu.layout_menu, menu)

  val layoutButton = menu?.findItem(R.id.action_switch_layout)
  // Calls code to set the icon based on the LinearLayoutManager of the RecyclerView
  setIcon(layoutButton)

  return true
}
```

Nie ma tu nic wyszukanego. Po nadmuchaniu układu dzwonisz, `setIcon()`aby upewnić się, że ikona jest poprawna, na podstawie układu. Metoda zwraca a `Boolean`— wracasz `true`tutaj, ponieważ chcesz utworzyć menu opcji.

2. Zaimplementuj, jak pokazano `onOptionsItemSelected`, z kilkoma dodatkowymi wierszami kodu.

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
  return when (item.itemId) {
     R.id.action_switch_layout -> {
         // Sets isLinearLayoutManager (a Boolean) to the opposite value
         isLinearLayoutManager = !isLinearLayoutManager
         // Sets layout and icon
         chooseLayout()
         setIcon(item)

         return true
     }
     //  Otherwise, do nothing and use the core event handling

     // when clauses require that all possible paths be accounted for explicitly,
     //  for instance both the true and false cases if the value is a Boolean,
     //  or an else to catch all unhandled cases.
     else -> super.onOptionsItemSelected(item)
```

```
    }
}
```

Jest to wywoływane za każdym razem, gdy element menu zostanie dotknięty, więc musisz upewnić się, że sprawdziłeś, który element menu został dotknięty. Używasz `when`powyższego oświadczenia. Jeśli `id`pasuje do `action_switch_layout`pozycji menu, negujesz wartość `isLinearLayoutManager`. Następnie zadzwoń `chooseLayout()`i `setIcon()`odpowiednio zaktualizuj interfejs użytkownika.

Jeszcze jedno, zanim uruchomisz aplikację. Ponieważ menedżer układu i adapter są teraz ustawione w `chooseLayout()`, należy zastąpić ten kod, `onCreate()`aby wywołać nową metodę. `onCreate()`po zmianie powinien wyglądać następująco.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    val binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    recyclerView = binding.recyclerView
    // Sets the LinearLayoutManager of the recyclerview
    chooseLayout()
}
```

Teraz uruchom swoją aplikację i powinieneś być w stanie przełączać się między widokami listy i siatki za pomocą przycisku menu.

10. Kod rozwiązania

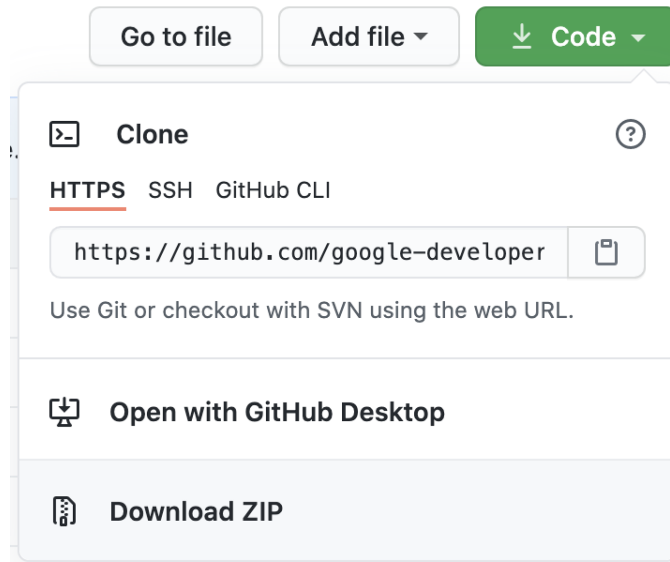
Kod rozwiązania dla tego ćwiczenia z programowania znajduje się w poniższym projekcie:

Adres URL kodu rozwiązania: https://github.com/google-developer-training/android-basics-kotlin-words-app/tree/activities

Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

Pobierz kod

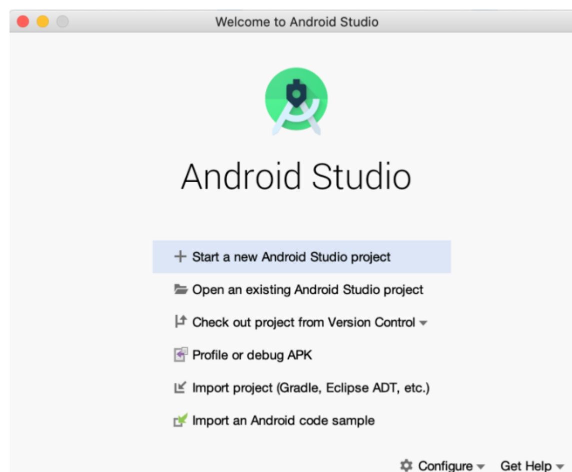
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



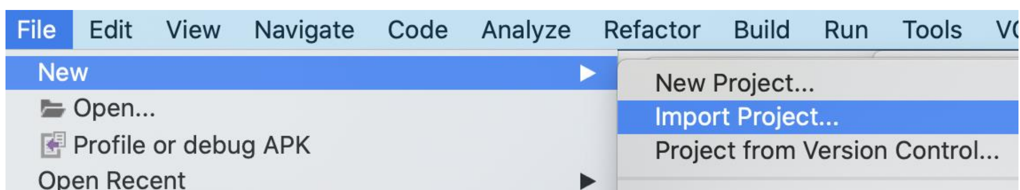
3. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
4. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
5. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

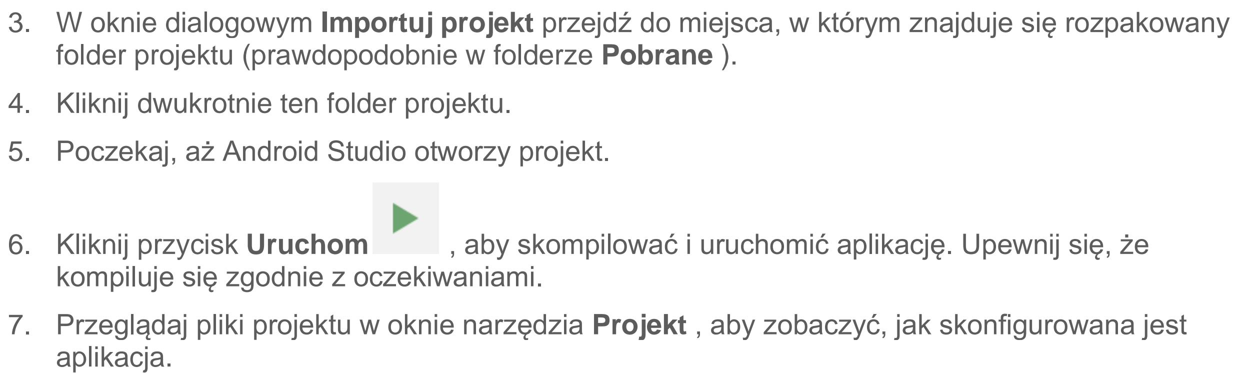
Otwórz projekt w Android Studio

1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz istniejący projekt Android Studio.**



Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Nowy** > **Importuj projekt** .



3. W oknie dialogowym **Importuj projekt** przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.
6. Kliknij przycisk **Uruchom**  , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.
7. Przeglądaj pliki projektu w oknie narzędzia **Projekt** , aby zobaczyć, jak skonfigurowana jest aplikacja.

11. Podsumowanie

- Jawne intencje służą do nawigowania do określonych działań w Twojej aplikacji.
- Niejawne intencje odpowiadają konkretnym działaniom (takim jak otwarcie linku lub udostępnienie obrazu) i pozwalają systemowi określić, jak zrealizować intencję.
- Opcje menu umożliwiają dodawanie przycisków i menu do paska aplikacji.
- Obiekty towarzyszące zapewniają sposób kojarzenia stałych wielokrotnego użytku z typem, a nie z wystąpieniem tego typu.

Aby wykonać zamiar:

- Uzyskaj odniesienie do kontekstu.
- Utwórz `Intent`obiekt dostarczający rodzaj działania lub intencji (w zależności od tego, czy jest to jawne, czy niejawne).
- Przekaż potrzebne dane dzwoniąc `putExtra()`.
- Zadzwoń `startActivity()`przechodząc w `intent`obiekcie.

12. Dowiedz się więcej

- Intencje i filtry intencji
- Powszechnie używane intencje
- Tworzenie menu opcji za pomocą XML
- Interfejsy w Kotlin
- Interfejs CharSequence
- Null Safety w Kotlin
- Wyrażenia i deklaracje obiektów w Kotlin
- Jednotonowy wzór
- Kontrola przepływu w Kotlin

Etapy cyklu życia działalności

1. Witamy!

Wstęp

W tym ćwiczeniu z programowania dowiesz się więcej o podstawowej części Androida: *działaniu* . Cykl *życia działania* to zestaw stanów, w jakich może znajdować się działanie podczas swojego życia. Cykl życia rozciąga się od momentu początkowego utworzenia działania do momentu jego zniszczenia, a system odzyskuje zasoby tego działania. Gdy użytkownik przechodzi między działaniami w Twojej aplikacji (oraz do iz aplikacji), te działania przechodzą między różnymi stanami w cyklu życia działania.

Jako programista Androida musisz zrozumieć cykl życia aktywności. Jeśli Twoje działania nie reagują prawidłowo na zmiany stanu cyklu życia, Twoja aplikacja może generować dziwne błędy, mylące zachowanie użytkowników lub zużywać zbyt wiele zasobów systemu Android. Zrozumienie cyklu życia Androida i prawidłowe reagowanie na zmiany stanu cyklu życia ma kluczowe znaczenie dla bycia dobrym obywatelem Androida.

Co powinieneś już wiedzieć

- Czym jest aktywność i jak ją utworzyć w swojej aplikacji.
- Co `onCreate()`robi metoda działania i rodzaj operacji, które są wykonywane w tej metodzie.

Czego się nauczysz

- Jak wydrukować informacje logowania do Logcata.
- Podstawy `Activity`cyklu życia i wywołania zwrotne, które są wywoływane, gdy aktywność przechodzi między stanami.
- Jak przesłonić metody wywołania zwrotnego cyklu życia, aby wykonywać operacje w różnych momentach cyklu życia działania.

Co zrobisz

- Zmodyfikuj aplikację startową o nazwie DessertClicker, aby dodać informacje rejestrowania wyświetlane w Logcat.
- Zastąp metody wywołania zwrotnego cyklu życia i rejestruj zmiany stanu aktywności.
- Uruchom aplikację i zanotuj informacje rejestrowania, które pojawiają się, gdy działanie jest rozpoczynane, zatrzymywane i wznawiane.
- Zaimplementuj `onSaveInstanceState()`metodę, aby zachować dane aplikacji, które mogą zostać utracone w przypadku zmiany konfiguracji urządzenia. Dodaj kod, aby przywrócić te dane po ponownym uruchomieniu aplikacji.

2. Przegląd aplikacji

W tym ćwiczeniu z programowania pracujesz z aplikacją startową o nazwie DessertClicker. W tej aplikacji za każdym razem, gdy użytkownik kliknie deser na ekranie, aplikacja „kupuje" deser dla

użytkownika. Aplikacja aktualizuje wartości w układzie dla liczby zakupionych deserów i całkowitej kwoty wydanej przez użytkownika.

Ta aplikacja zawiera kilka błędów związanych z cyklem życia Androida: Na przykład w pewnych okolicznościach aplikacja resetuje wartości deseru do 0. Zrozumienie cyklu życia Androida pomoże Ci zrozumieć, dlaczego występują te problemy i jak je naprawić.

Pobierz aplikację startową

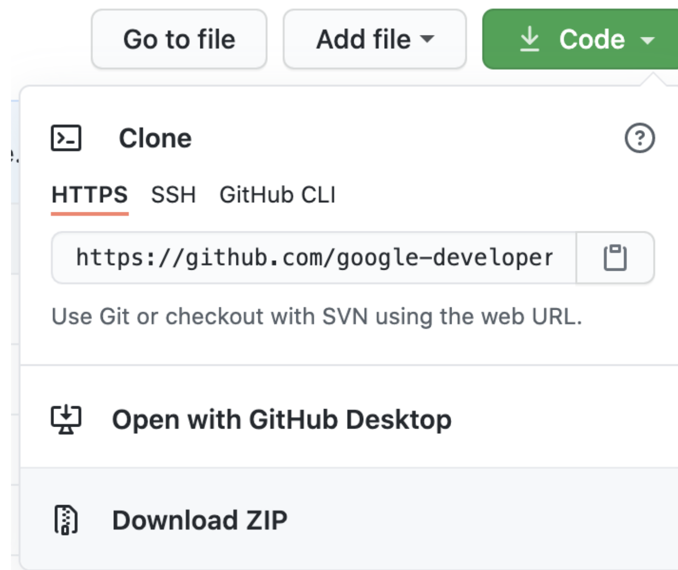
Pobierz kod startowy DessertClicker i otwórz go w Android Studio.

Jeśli używasz kodu startowego z GitHub, pamiętaj, że nazwa folderu to `android-basics-kotlin-dessert-clicker-app-starter`. Wybierz ten folder podczas otwierania projektu w Android Studio.

Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

Pobierz kod

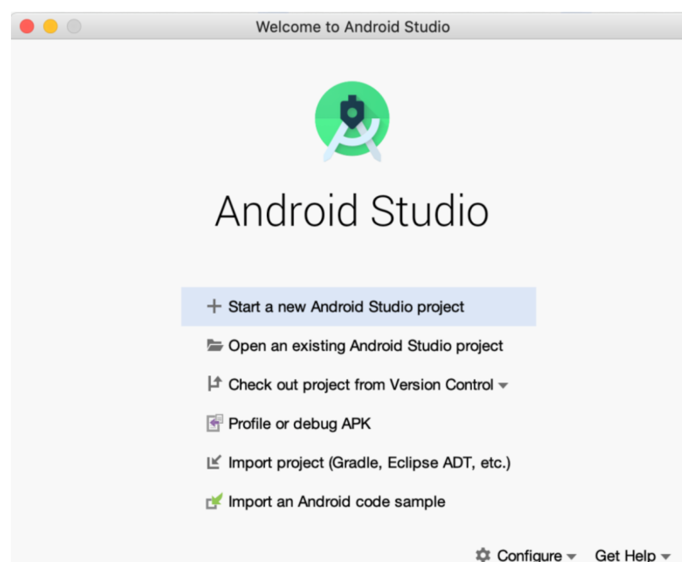
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



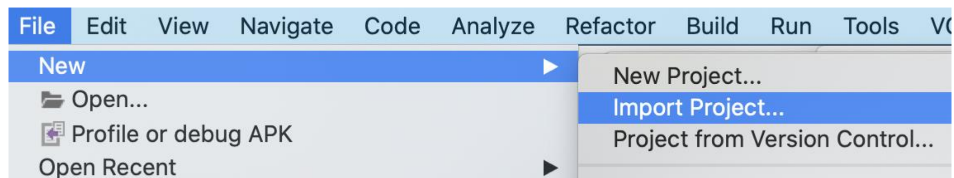
3. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
4. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
5. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

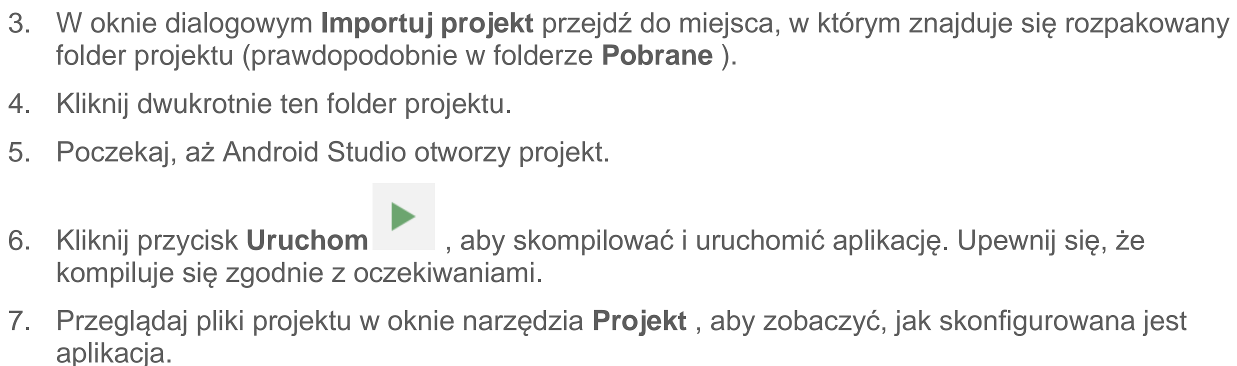
Otwórz projekt w Android Studio

1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz istniejący projekt Android Studio.**



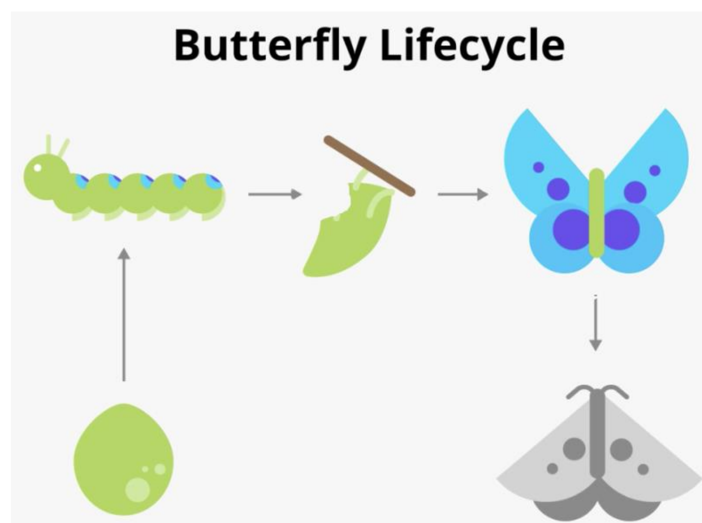
Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Nowy** > **Importuj projekt** .



3. W oknie dialogowym **Importuj projekt** przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.
6. Kliknij przycisk **Uruchom**  , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.
7. Przeglądaj pliki projektu w oknie narzędzia **Projekt** , aby zobaczyć, jak skonfigurowana jest aplikacja.

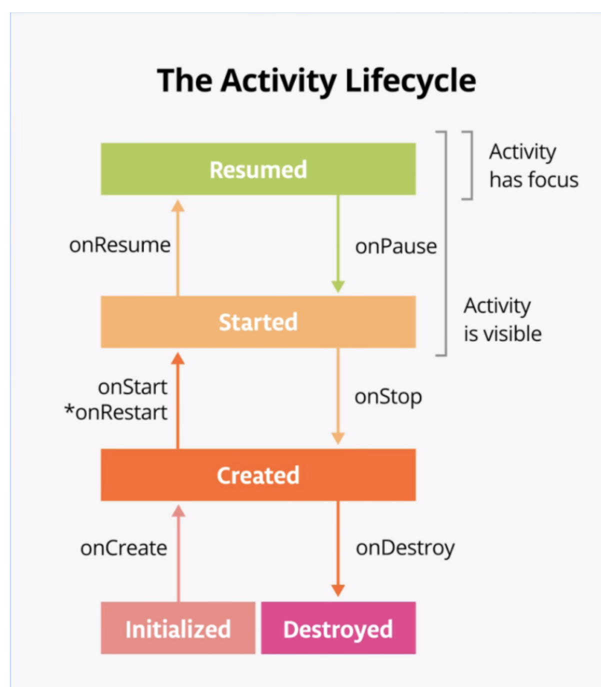
3. Poznaj metody cyklu życia i dodaj podstawowe rejestrowanie

Każde działanie ma tak zwany *cykl życia* . Jest to aluzja do cykli życiowych roślin i zwierząt, takich jak cykl życiowy tego motyla — różne stany motyla pokazują jego wzrost od narodzin przez w pełni ukształtowaną dorosłość aż do śmierci.



Podobnie cykl życia działania składa się z różnych stanów, przez które może przejść działanie, od pierwszego zainicjowania działania do ostatecznego zniszczenia i odzyskania pamięci przez system. Gdy użytkownik uruchamia aplikację, nawiguje między działaniami, nawiguje wewnątrz i na zewnątrz aplikacji, stan działania zmienia się. Poniższy diagram przedstawia wszystkie stany cyklu życia aktywności. Jak wskazują ich nazwy, stany te reprezentują stan działalności.

Często chcesz zmienić pewne zachowanie lub uruchomić kod, gdy zmieni się stan cyklu życia działania. Dlatego `Activity`sama klasa i wszelkie podklasy, `Activity`takie jak `AppCompatActivity`, implementują zestaw metod wywołania zwrotnego cyklu życia. System Android wywołuje te wywołania zwrotne, gdy działanie przechodzi z jednego stanu do drugiego, i można zastąpić te metody we własnych działaniach, aby wykonywać zadania w odpowiedzi na te zmiany stanu cyklu życia. Poniższy diagram przedstawia stany cyklu życia wraz z dostępnymi wywołaniami zwrotnymi, które można zastąpić.



Ważne jest, aby wiedzieć, kiedy te wywołania zwrotne są wywoływane i co robić w każdej metodzie wywołania zwrotnego. Ale oba te diagramy są złożone i mogą być mylące. W tym laboratorium kodowania, zamiast tylko czytać, co oznaczają poszczególne stany i wywołania zwrotne, wykonasz trochę pracy detektywistycznej i zbudujesz swoje zrozumienie tego, co się dzieje.

Krok 1: Sprawdź metodę onCreate() i dodaj logowanie

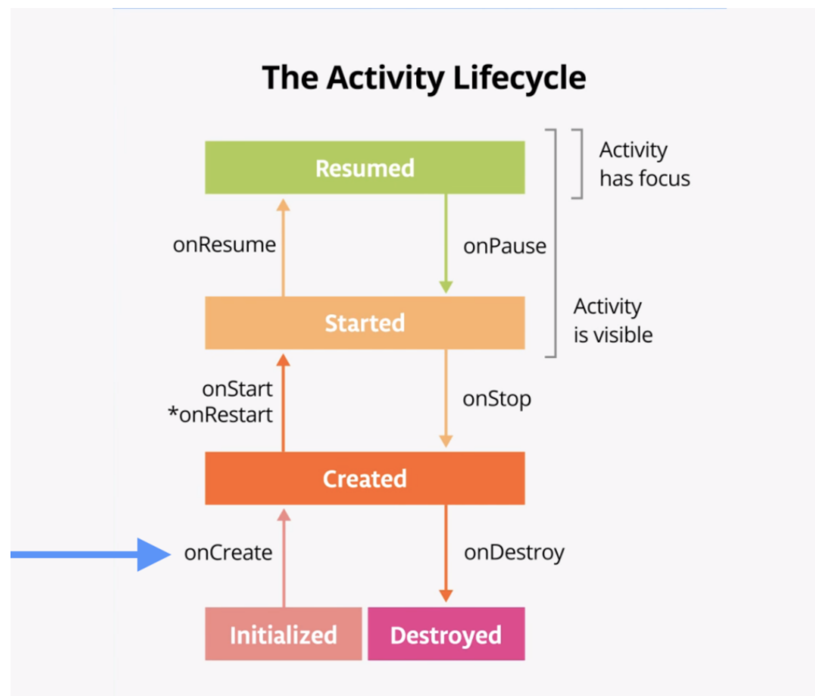
Aby dowiedzieć się, co się dzieje z cyklem życia systemu Android, warto wiedzieć, kiedy wywoływane są różne metody cyklu życia. Pomoże Ci to znaleźć, gdzie w DessertClicker dzieje się coś złego.

Prostym sposobem na to jest użycie funkcji rejestrowania w systemie Android. Rejestrowanie umożliwia pisanie krótkich wiadomości do konsoli podczas działania aplikacji i umożliwia pokazanie, kiedy zostaną wyzwolone różne wywołania zwrotne.

1. Uruchom aplikację Dessert Clicker i dotknij kilka razy zdjęcie deseru. Zwróć uwagę, jak zmienia się wartość **Sprzedanych Deserów** i łączna kwota w dolarach.
2. Otwórz MainActivity.kti zbadaj onCreate()metodę dla tego ćwiczenia:

```
override fun onCreate(savedInstanceState: Bundle?) {
...
}
```

Na diagramie cyklu życia działania być może rozpoznałeś tę onCreate()metodę, ponieważ wcześniej używałeś tego wywołania zwrotnego. To jedyna metoda, którą musi wdrożyć każde działanie. Metoda onCreate()polega na tym, że powinieneś wykonać jednorazowe inicjalizacje dla swojej aktywności. Na przykład, onCreate()gdy nadmuchujesz układ, zdefiniuj detektory kliknięć lub skonfiguruj powiązanie widoku.



Metoda onCreate()cyklu życia jest wywoływana jednorazowo, zaraz po zainicjowaniu aktywności (kiedy nowy Activityobiekt jest tworzony w pamięci). Po onCreate()wykonaniu czynność jest uważana za *utworzoną* .

Uwaga: Gdy nadpisujesz onCreate()metodę, musisz wywołać implementację nadklasy, aby zakończyć tworzenie Aktywności, więc w jej obrębie musisz natychmiast wywołać super.onCreate(). To samo dotyczy innych metod wywołania zwrotnego cyklu życia.

3. W onCreate()metodzie zaraz po wywołaniu do super.onCreate()dodaj następujący wiersz:

```
Log.d("MainActivity", "onCreate Called")
```

4. W razie potrzeby zaimportuj `Log`klasę (naciśnij `Alt+Enter`lub `Option+Enter`na Macu i wybierz **Importuj**). Jeśli włączyłeś automatyczne importowanie, powinno to nastąpić automatycznie.

```
import android.util.Log
```

Klasa `Log`zapisuje komunikaty do **Logcat** . Logcat to **konsola** do rejestrowania komunikatów. Tutaj pojawiają się komunikaty z Androida dotyczące Twojej aplikacji, w tym komunikaty, które wprost wysyłasz do dziennika za pomocą `Log.d()`metody lub innych `Log`metod klasy.

To polecenie składa się z trzech części:

- Priorytet komunikatu dziennika, czyli stopień ważności komunikatu . W takim przypadku `Log.d()`metoda zapisuje komunikat debugowania. Inne metody w `Log`klasie obejmują `Log.i()`komunikaty informacyjne, `Log.e()`błędy, `Log.w()`ostrzeżenia lub `Log.v()`pełne komunikaty.
- *Znacznik* dziennika (pierwszy parametr), w tym przypadku `"MainActivity"`. Znacznik jest ciągiem, który ułatwia znajdowanie komunikatów dziennika w Logcat. Znacznik jest zazwyczaj nazwą klasy.
- Rzeczywisty *komunikat* dziennika (drugi parametr) to krótki ciąg, którym w tym przypadku jest `"onCreate called"`.

Uwaga: Dobrą konwencją jest deklarowanie stałej TAG w swojej klasie:

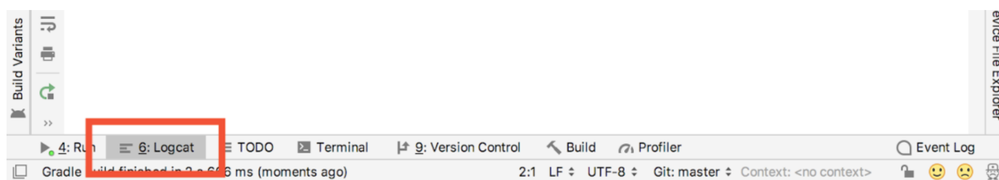
```
const val TAG = "MainActivity"
```

i użyj tego w kolejnych wywołaniach metod log, jak poniżej:

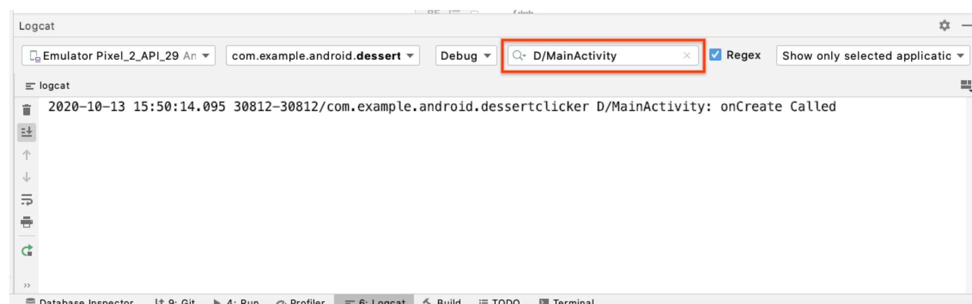
```
Log.d(TAG, "onCreate Called")
```

Stała *czasu kompilacji* to wartość, która się nie zmienia. Użyj `const`przed deklaracją zmiennej, aby oznaczyć ją jako stałą czasu kompilacji.

5. Skompiluj i uruchom aplikację DessertClicker. Po dotknięciu deseru nie widać żadnych różnic w zachowaniu w aplikacji. W Android Studio na dole ekranu kliknij kartę **Logcat** .



6. W oknie **Logcat** wpisz `D/MainActivity`w polu wyszukiwania.

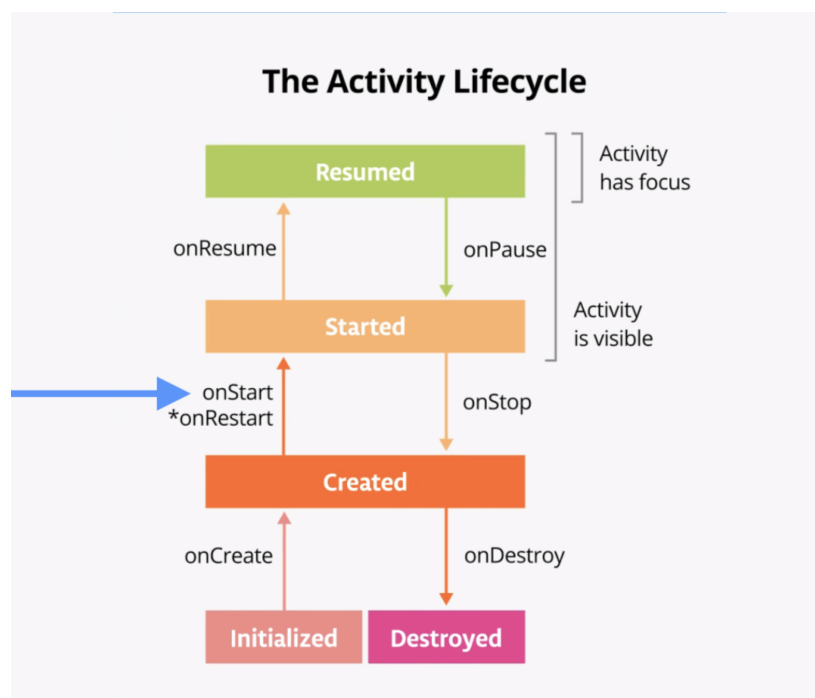


Logcat może zawierać wiele wiadomości, z których większość nie jest dla Ciebie przydatna. Wpisy Logcat można filtrować na wiele sposobów, ale wyszukiwanie jest najłatwiejsze. Ponieważ użyłeś go MainActivityjako tagu dziennika w kodzie, możesz użyć tego tagu do filtrowania dziennika. Dodanie D/na początku oznacza, że jest to komunikat debugowania, stworzony przez Log.d().

Twoja wiadomość dziennika zawiera datę i godzinę, nazwę pakietu (com.example.android.dessertclicker), tag dziennika (z D/na początku) i rzeczywistą wiadomość. Ponieważ ten komunikat pojawia się w dzienniku, wiesz, że onCreate()został wykonany.

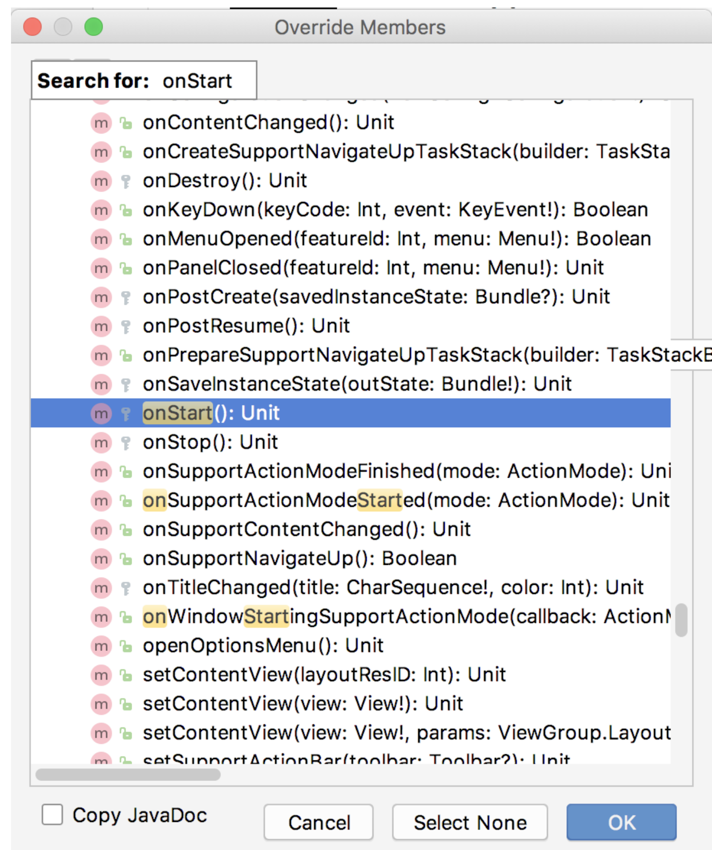
Krok 2: Zaimplementuj metodę onStart()

Metoda onStart()cyklu życia jest wywoływana zaraz po onCreate(). Po onStart()biegach Twoja aktywność jest widoczna na ekranie. W przeciwieństwie do onCreate(), który jest wywoływany tylko raz w celu zainicjowania działania, onStart()można go wywoływać wiele razy w cyklu życia działania.



Zauważ, że onStart()jest sparowany z odpowiednią onStop()metodą cyklu życia. Jeśli użytkownik uruchomi aplikację, a następnie powróci do ekranu głównego urządzenia, aktywność zostanie zatrzymana i nie będzie już widoczna na ekranie.

1. W Android Studio, z MainActivity.ktotwartym i kursorem w MainActivityklasie, wybierz **Kod > Zastąp metody** lub naciśnij Control+o(Command+ona Macu). Pojawi się okno dialogowe z ogromną listą wszystkich metod, które można przesłonić w tej klasie.



2. Zacznij wpisywać onStart, aby wyszukać właściwą metodę. Aby przewinąć do następnego pasującego elementu, użyj strzałki w dół. Wybierz onStart()z listy i kliknij **OK** , aby wstawić kod nadpisania standardowego. Kod wygląda tak:

```
override fun onStart() {
   super.onStart()
}
```

3. Dodaj następującą stałą na najwyższym poziomie MainActivity.kt, czyli powyżej deklaracji klasy,class MainActivity.

```
const val TAG = "MainActivity"
```

4. Wewnątrz onStart()metody dodaj komunikat dziennika:

```
override fun onStart() {
   super.onStart()
   Log.d(TAG, "onStart Called")
}
```

5. Skompiluj i uruchom aplikację DessertClicker, a następnie otwórz okienko **Logcat** . Wpisz D/MainActivityw polu wyszukiwania, aby przefiltrować dziennik. Zwróć uwagę, że obie metody onCreate()i onStart()zostały wywołane jedna po drugiej, a Twoja aktywność jest widoczna na ekranie.
6. Naciśnij przycisk Początek na urządzeniu, a następnie użyj ekranu ostatnich lat, aby powrócić do aktywności. Zauważ, że aktywność wznawia się w miejscu, w którym została przerwana, ze

wszystkimi tymi samymi wartościami, co onStart()jest rejestrowane po raz drugi w Logcat. Zauważ też, że onCreate()metoda zwykle nie jest wywoływana ponownie.

```
16:19:59.125 31107-31107/com.example.android.dessertclicker D/MainActivity: onCreate Called
16:19:59.372 31107-31107/com.example.android.dessertclicker D/MainActivity: onStart Called
16:20:11.319 31107-31107/com.example.android.dessertclicker D/MainActivity: onStart Called
```

Uwaga: eksperymentując z urządzeniem i obserwując wywołania zwrotne cyklu życia, możesz zauważyć nietypowe zachowanie podczas obracania urządzenia. Dowiesz się o tym zachowaniu w dalszej części tego ćwiczenia z programowania.

Krok 3: Dodaj więcej instrukcji dziennika

W tym kroku zaimplementujesz rejestrowanie dla wszystkich innych metod cyklu życia.

1. Zastąp pozostałe metody cyklu życia w swoim MainActivityi dodaj instrukcje dziennika dla każdej z nich. Oto kod:

```
override fun onResume() {
   super.onResume()
   Log.d(TAG, "onResume Called")
}

override fun onPause() {
   super.onPause()
   Log.d(TAG, "onPause Called")
}

override fun onStop() {
   super.onStop()
   Log.d(TAG, "onStop Called")
}

override fun onDestroy() {
   super.onDestroy()
   Log.d(TAG, "onDestroy Called")
}

override fun onRestart() {
   super.onRestart()
   Log.d(TAG, "onRestart Called")
}
```

2. Skompiluj i uruchom ponownie DessertClicker i zbadaj Logcat. Tym razem zauważ, że oprócz onCreate()i onStart(), istnieje komunikat dziennika dla onResume()wywołania zwrotnego cyklu życia.

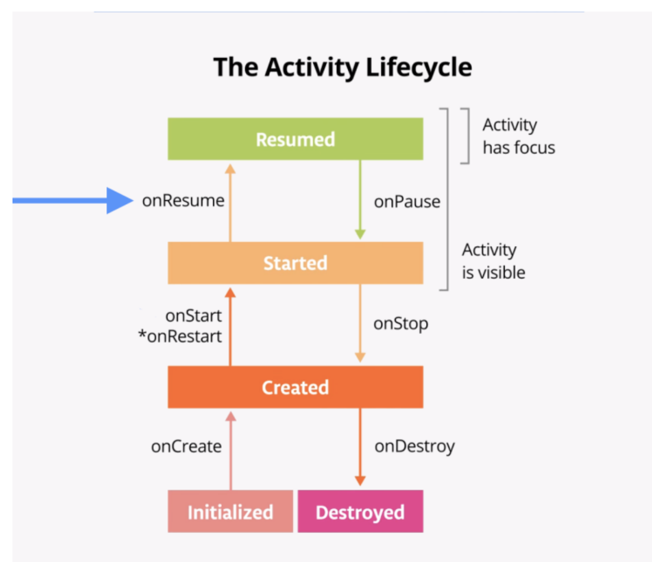
```
2020-10-16 10:27:33.244 22064-22064/com.example.android.dessertclicker D/MainActivity: onCreate Called
```

2020-10-16 10:27:33.453 22064-22064/com.example.android.dessertclicker D/MainActivity: onStart Called
2020-10-16 10:27:33.454 22064-22064/com.example.android.dessertclicker D/MainActivity: onResume Called

Gdy działanie zaczyna się od zera, zobaczysz wszystkie trzy wywołania zwrotne cyklu życia wywoływane w kolejności:

- onCreate()aby stworzyć aplikację.
- onStart()aby go uruchomić i uwidocznić na ekranie.
- onResume()aby skoncentrować się na aktywności i przygotować ją do interakcji użytkownika.

Pomimo nazwy onResume()metoda jest wywoływana podczas uruchamiania, nawet jeśli nie ma nic do wznowienia.



4. Poznaj przypadki użycia w cyklu życia

Teraz, gdy aplikacja DessertClicker jest skonfigurowana do rejestrowania, możesz rozpocząć korzystanie z aplikacji na różne sposoby i zbadać, w jaki sposób wywołania zwrotne cyklu życia są wyzwalane w odpowiedzi na te zastosowania.

Przypadek użycia 1: Otwieranie i zamykanie ćwiczenia

Zaczynasz od najbardziej podstawowego przypadku użycia, którym jest uruchomienie aplikacji po raz pierwszy, a następnie jej całkowite zamknięcie.

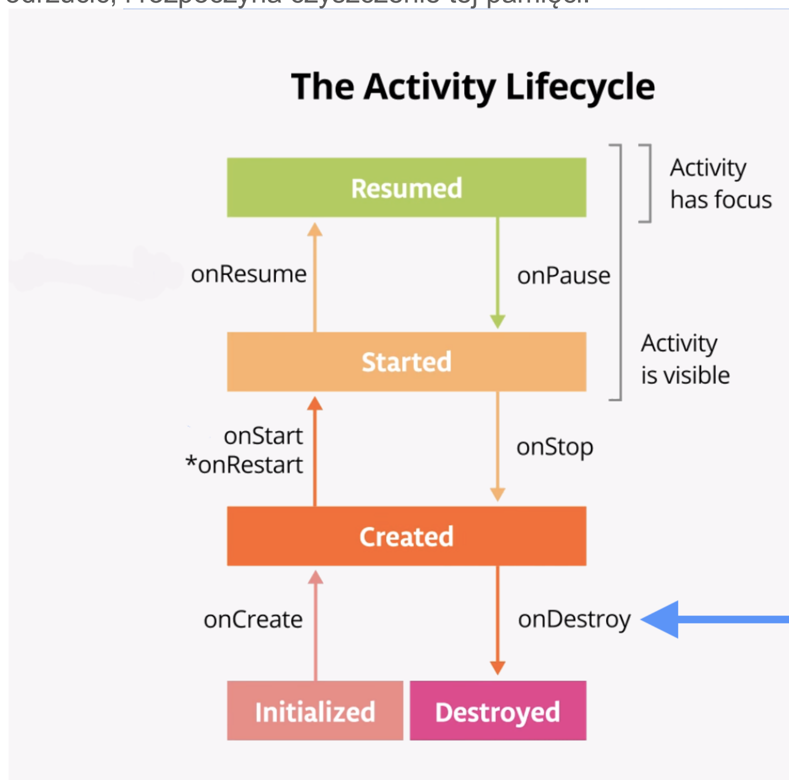
1. Skompiluj i uruchom aplikację DessertClicker, jeśli jeszcze nie jest uruchomiona. Jak widać, onCreate()wywołania zwrotne onStart(), i onResume()są wywoływane, gdy czynność rozpoczyna się po raz pierwszy.

2020-10-16 10:27:33.244 22064-22064/com.example.android.dessertclicker D/MainActivity: onCreate Called

2020-10-16 10:27:33.453 22064-22064/com.example.android.dessertclicker D/MainActivity: onStart Called
2020-10-16 10:27:33.454 22064-22064/com.example.android.dessertclicker D/MainActivity: onResume Called

2. Stuknij ciastko kilka razy.
3. Stuknij przycisk **Wstecz** na urządzeniu. Zauważ w Logcat, że onPause(), onStop()i onDestroy()są wywoływane w tej kolejności.

2020-10-16 10:31:53.850 22064-22064/com.example.android.dessertclicker D/MainActivity: onPause Called
2020-10-16 10:31:54.620 22064-22064/com.example.android.dessertclicker D/MainActivity: onStop Called
2020-10-16 10:31:54.622 22064-22064/com.example.android.dessertclicker D/MainActivity: onDestroy Called
W takim przypadku użycie przycisku **Wstecz** powoduje całkowite zamknięcie aktywności (i aplikacji). Wykonanie onDestroy()metody oznacza, że działanie zostało całkowicie zamknięte i może zostać usunięte. *Odśmiecanie* odnosi się do automatycznego czyszczenia obiektów, których już nie będziesz używać. Po onDestroy()wywołaniu system wie, że te zasoby można odrzucić, i rozpoczyna czyszczenie tej pamięci.



Twoja aktywność może również zostać całkowicie zamknięta, jeśli kod ręcznie wywoła finish()metodę aktywności lub jeśli użytkownik wymusza zamknięcie aplikacji. (Na przykład użytkownik może wymusić zamknięcie lub zamknięcie aplikacji na ekranie ostatnich). System Android może również sam wyłączyć Twoją aktywność, jeśli Twoja aplikacja nie była wyświetlana na ekranie przez długi czas. Android robi to, aby oszczędzać baterię i umożliwić innym aplikacjom korzystanie z zasobów Twojej aplikacji.

4. Wróć do aplikacji DessertClicker, znajdując wszystkie otwarte aplikacje na ekranie Przegląd . (Zauważ, że jest to również znane jako ekran Ostatnie lub ostatnie aplikacje.) Oto Logcat:

2020-10-16 10:31:54.622 22064-22064/com.example.android.dessertclicker D/MainActivity: onDestroy Called

2020-10-16 10:38:00.733 22064-22064/com.example.android.dessertclicker D/MainActivity: onCreate Called
2020-10-16 10:38:00.787 22064-22064/com.example.android.dessertclicker D/MainActivity: onStart Called
2020-10-16 10:38:00.788 22064-22064/com.example.android.dessertclicker D/MainActivity: onResume Called

Działanie zostało zniszczone w poprzednim kroku, więc po powrocie do aplikacji system Android uruchamia nowe działanie i wywołuje metody onCreate(), onStart()i onResume(). Zwróć uwagę, że żaden z dzienników DessertClicker z poprzedniego działania nie został zachowany.

Uwaga: Kluczową kwestią jest to, że onCreate()i onDestroy()są wywoływane tylko raz w okresie istnienia pojedynczej instancji aktywności: onCreate()w celu zainicjowania aplikacji po raz pierwszy i onDestroy()oczyszczenia zasobów używanych przez aplikację.

Metoda onCreate()jest ważnym krokiem; to jest miejsce, w którym odbywa się cała twoja pierwsza inicjalizacja, gdzie po raz pierwszy konfigurujesz układ, rozszerzając go i gdzie inicjujesz zmienne.

Przypadek użycia 2: Nawigacja z powrotem do działania i z powrotem

Teraz, po uruchomieniu aplikacji i całkowitym jej zamknięciu, widzisz większość stanów cyklu życia, gdy działanie jest tworzone po raz pierwszy. Widziałeś również wszystkie stany cyklu życia, przez które aktywność przechodzi, gdy zostaje całkowicie zamknięta i zniszczona. Ale gdy użytkownicy wchodzą w interakcję ze swoimi urządzeniami z Androidem, przełączają się między aplikacjami, wracają do domu, uruchamiają nowe aplikacje i obsługują przerwy spowodowane innymi czynnościami, takimi jak rozmowy telefoniczne.

Twoja aktywność nie kończy się całkowicie za każdym razem, gdy użytkownik odchodzi od tej aktywności:

- Gdy Twoja aktywność nie jest już widoczna na ekranie, nazywa się to umieszczeniem jej w *tle* . (Odwrotnością tego jest sytuacja, gdy aktywność jest na *pierwszym planie* lub na ekranie).
- Gdy użytkownik wróci do Twojej aplikacji, ta sama aktywność zostanie ponownie uruchomiona i znów będzie widoczna. Ta część cyklu życia nazywana jest *widocznym* cyklem życia aplikacji.

Gdy aplikacja działa w tle, zazwyczaj nie powinna działać aktywnie, aby oszczędzać zasoby systemowe i żywotność baterii. Używasz Activitycyklu życia i jego wywołań zwrotnych, aby wiedzieć, kiedy Twoja aplikacja przechodzi w tło, dzięki czemu możesz wstrzymać wszelkie trwające operacje. Następnie ponownie uruchamiasz operacje, gdy aplikacja pojawi się na pierwszym planie.

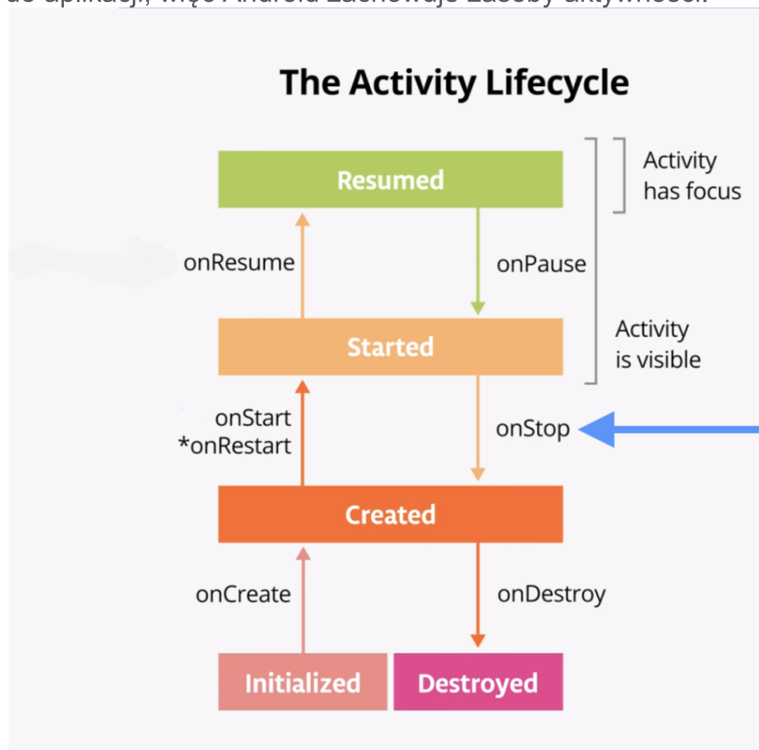
W tym kroku przyjrzysz się cyklowi życia aktywności, gdy aplikacja przechodzi w tło i powraca na pierwszy plan.

1. Po uruchomieniu aplikacji DessertClicker kliknij kilka razy ciastko.
2. Naciśnij przycisk **Home** na swoim urządzeniu i obserwuj Logcat w Android Studio. Powrót do ekranu głównego umieszcza aplikację w tle, zamiast całkowicie ją wyłączać. Zauważ, że onPause()metoda i onStop()metody są wywoływane, ale onDestroy()nie są.

2020-10-16 10:41:05.383 22064-22064/com.example.android.dessertclicker D/MainActivity: onPause Called

2020-10-16 10:41:05.966 22064-22064/com.example.android.dessertclicker D/MainActivity: onStop Called

Kiedy onPause()jest wywoływana, aplikacja nie jest już skoncentrowana. Po onStop()aplikacji aplikacja nie jest już widoczna na ekranie. Chociaż czynność została zatrzymana, Activityobiekt nadal znajduje się w pamięci, w tle. Działalność nie została zniszczona. Użytkownik może wrócić do aplikacji, więc Android zachowuje zasoby aktywności.



3. Użyj ekranu z ostatnimi, aby wrócić do aplikacji. Zwróć uwagę w Logcat, że aktywność jest ponownie uruchamiana za pomocą onRestart()i onStart(), a następnie wznawiana za pomocą onResume().

   2020-10-16 10:42:18.144 22064-22064/com.example.android.dessertclicker D/MainActivity: onRestart Called
   2020-10-16 10:42:18.158 22064-22064/com.example.android.dessertclicker D/MainActivity: onStart Called
   2020-10-16 10:42:18.158 22064-22064/com.example.android.dessertclicker D/MainActivity: onResume Called

   Gdy działanie powraca na pierwszy plan, onCreate()metoda nie jest wywoływana ponownie. Obiekt aktywności nie został zniszczony, więc nie trzeba go ponownie tworzyć. Zamiast onCreate()metody onRestart()wywoływana jest metoda. Zauważ, że tym razem, gdy aktywność powraca na pierwszy plan, numer **Desserts Sold** zostaje zachowany.

4. Uruchom co najmniej jedną aplikację inną niż DessertClicker, aby urządzenie miało kilka aplikacji na ekranie najnowszych.
5. Wyświetl ekran ostatnich wydarzeń i otwórz inną ostatnią aktywność. Następnie wróć do ostatnich aplikacji i przenieś DessertClicker z powrotem na pierwszy plan.

   Zauważ, że widzisz tutaj te same wywołania zwrotne w Logcat, co po naciśnięciu przycisku Home. onPause()i onStop()są wywoływane, gdy aplikacja przechodzi w tło, a następnie onRestart(), onStart()i onResume()po powrocie.

Uwaga: Ważnym punktem jest to, że onStart()i onStop()są wywoływane wielokrotnie, gdy użytkownik przechodzi do działania i z niego.

Te metody są wywoływane, gdy aplikacja zostanie zatrzymana i przeniesiona do tła lub gdy aplikacja zostanie ponownie uruchomiona po powrocie na pierwszy plan. Jeśli w takich przypadkach musisz wykonać jakąś pracę w swojej aplikacji, zastąp odpowiednią metodę wywołania zwrotnego cyklu życia.

Więc co z onRestart()? Metoda onRestart()jest podobna do onCreate(). Albo jest wywoływana, onCreate()albo onRestart()jest wywoływana, zanim aktywność stanie się widoczna. Metoda onCreate()jest wywoływana tylko po raz pierwszy, a onRestart()następnie jest wywoływana. Metoda onRestart()to miejsce na umieszczenie kodu, który chcesz wywołać tylko wtedy, gdy Twoja aktywność **nie** jest uruchamiana po raz pierwszy.

Przypadek użycia 3: Częściowo ukryj czynność

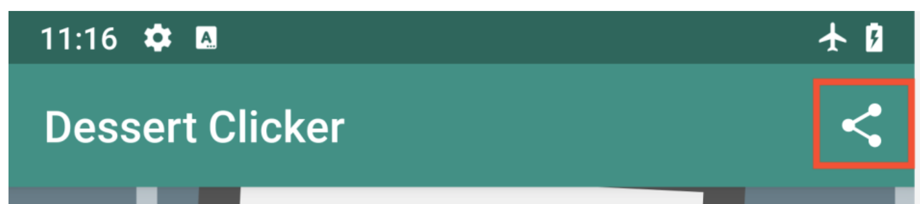
Dowiedziałeś się, że po uruchomieniu aplikacji i onStart()wywołaniu, aplikacja staje się widoczna na ekranie. Gdy aplikacja zostanie wznowiona i onResume()wywołana, aplikacja staje się skupiona na użytkowniku, co oznacza, że użytkownik może wchodzić w interakcję z aplikacją. Część cyklu życia, w której aplikacja jest w pełni widoczna na ekranie i skupia się na użytkowniku, nazywa się *interaktywnym* cyklem życia.

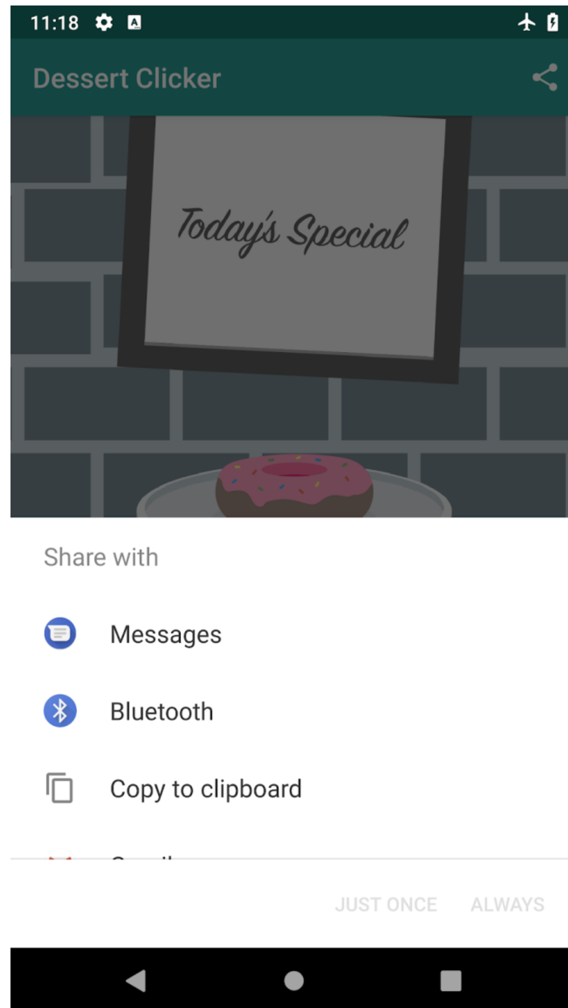
Gdy aplikacja przechodzi w tło, fokus jest tracony po onPause(), a aplikacja nie jest już widoczna po onStop().

Różnica między fokusem a widocznością jest ważna, ponieważ aktywność może być *częściowo* widoczna na ekranie, ale nie ma fokusu użytkownika. W tym kroku przyjrzysz się jednemu przypadkowi, w którym aktywność jest częściowo widoczna, ale nie jest skoncentrowana na użytkowniku.

1. Po uruchomieniu aplikacji DessertClicker kliknij przycisk **Udostępnij** w prawym górnym rogu ekranu.

   Aktywność udostępniania pojawia się w dolnej połowie ekranu, ale aktywność jest nadal widoczna w górnej połowie.





2. Sprawdź Logcata i zauważ, że tylko onPause()został wywołany.

   2020-10-16 11:00:53.857 22064-22064/com.example.android.dessertclicker D/MainActivity: onPause Called

   W tym przypadku użycia onStop()nie jest wywoływany, ponieważ aktywność jest nadal częściowo widoczna. Ale działanie nie jest skupione na użytkowniku, a użytkownik nie może z nim wchodzić w interakcje — działanie „udostępniania", które jest na pierwszym planie, ma fokus na użytkownika.

   Dlaczego ta różnica jest ważna? Przerwa z onPause()zwykle trwa tylko chwilę, po czym następuje powrót do aktywności lub przejście do innej aktywności lub aplikacji. Zwykle chcesz aktualizować interfejs użytkownika, aby reszta aplikacji nie zawieszała się.

   Jakikolwiek kod zostanie uruchomiony onPause(), blokuje wyświetlanie innych rzeczy, więc zachowaj onPause()lekki kod. Na przykład, jeśli nadejdzie połączenie telefoniczne, kod onPause()wchodzący może opóźnić powiadomienie o połączeniu przychodzącym.

3. Kliknij poza oknem udostępniania, aby wrócić do aplikacji, i zwróć uwagę, że onResume()jest wywoływana.

   Oba onResume()i onPause()mają do czynienia z koncentracją. Metoda onResume()jest wywoływana, gdy aktywność ma fokus, i onPause()jest wywoływana, gdy aktywność traci fokus.

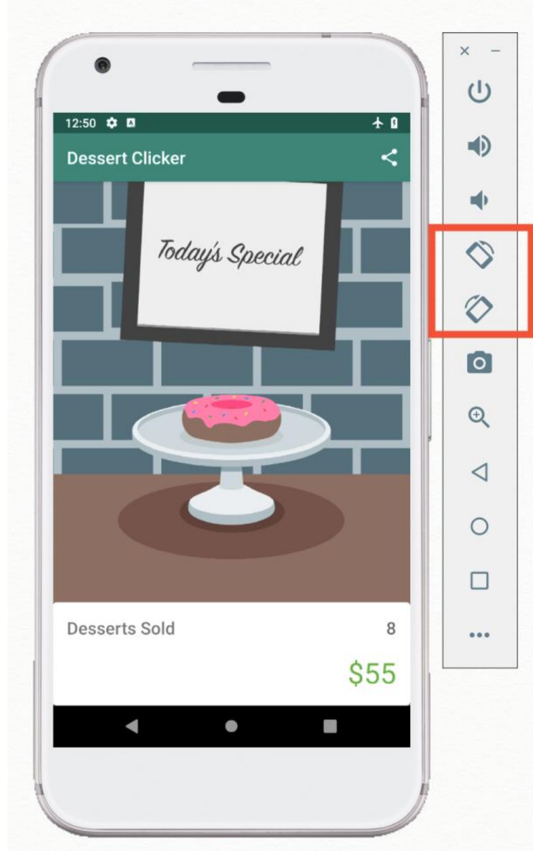
5. Poznaj zmiany w konfiguracji

Jest jeszcze jeden przypadek zarządzania cyklem życia działania, który jest ważny do zrozumienia: jak zmiany konfiguracji wpływają na cykl życia działań.

Zmiana *konfiguracji* ma miejsce, gdy stan urządzenia zmienia się tak radykalnie, że najłatwiejszym sposobem rozwiązania tej zmiany przez system jest całkowite zamknięcie i odbudowanie aktywności. Na przykład, jeśli użytkownik zmieni język urządzenia, cały układ może wymagać zmiany, aby uwzględnić różne kierunki tekstu i długości ciągów. Jeśli użytkownik podłączy urządzenie do stacji dokującej lub doda fizyczną klawiaturę, układ aplikacji może wymagać wykorzystania innego rozmiaru ekranu lub układu. A jeśli zmieni się orientacja urządzenia — jeśli urządzenie zostanie obrócone z pionowej na poziomą lub odwrotnie — układ może wymagać zmiany, aby pasował do nowej orientacji. Przyjrzyjmy się, jak zachowuje się aplikacja w tym scenariuszu.

Utrata danych przy obrocie urządzenia

1. Skompiluj i uruchom swoją aplikację, a następnie otwórz Logcat.
2. Obróć urządzenie lub emulator do trybu poziomego. Emulator można obracać w lewo lub w prawo za pomocą przycisków obracania lub za pomocą `Control`klawiszy strzałek i (`Command`i



klawiszy strzałek na komputerze Mac).
3. Sprawdź dane wyjściowe w Logcat. Przefiltruj dane wyjściowe na `MainActivity`.

2020-10-16 11:03:09.618 23206-23206/com.example.android.dessertclicker D/MainActivity: onCreate Called
2020-10-16 11:03:09.806 23206-23206/com.example.android.dessertclicker D/MainActivity: onStart Called
2020-10-16 11:03:09.808 23206-23206/com.example.android.dessertclicker D/MainActivity: onResume Called

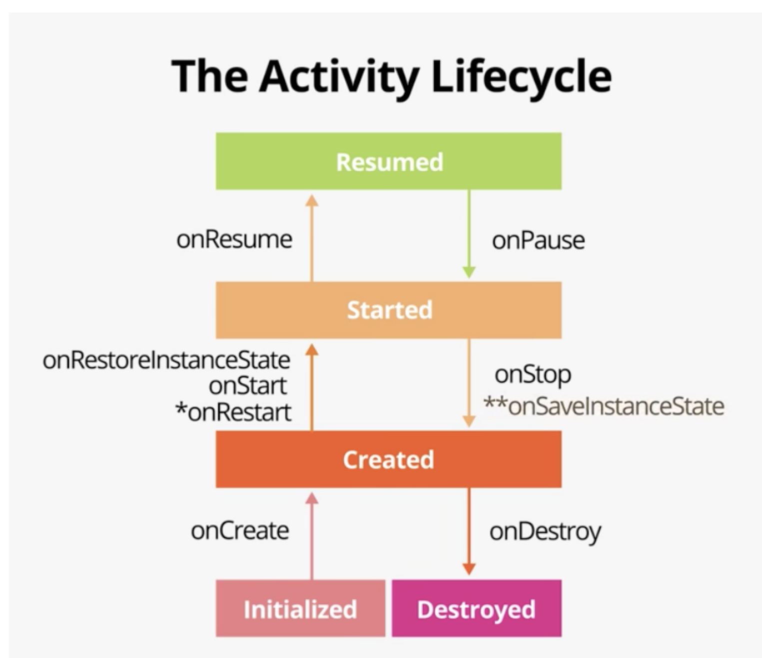
2020-10-16 11:03:24.488 23206-23206/com.example.android.dessertclicker D/MainActivity: onPause Called
2020-10-16 11:03:24.490 23206-23206/com.example.android.dessertclicker D/MainActivity: onStop Called
2020-10-16 11:03:24.493 23206-23206/com.example.android.dessertclicker D/MainActivity: onDestroy Called
2020-10-16 11:03:24.520 23206-23206/com.example.android.dessertclicker D/MainActivity: onCreate Called
2020-10-16 11:03:24.569 23206-23206/com.example.android.dessertclicker D/MainActivity: onStart Called

Zauważ, że gdy urządzenie lub emulator obraca ekran, system wywołuje wszystkie wywołania zwrotne cyklu życia, aby zamknąć aktywność. Następnie, gdy działanie jest ponownie tworzone, system wywołuje wszystkie wywołania zwrotne cyklu życia, aby rozpocząć działanie.

4. Gdy urządzenie zostanie obrócone, a działanie zostanie zamknięte i odtworzone, działanie zostanie uruchomione z wartościami domyślnymi — liczba sprzedanych deserów i przychody zostaną zresetowane do zera.

Użyj onSaveInstanceState(), aby zapisać dane pakietu

Metoda `onSaveInstanceState()`jest wywołaniem zwrotnym używanym do zapisywania wszelkich danych, które mogą być potrzebne w przypadku `Activity`zniszczenia. Na diagramie cyklu życia wywołania zwrotnego `onSaveInstanceState()`jest wywoływana po zatrzymaniu aktywności. Jest wywoływana za każdym razem, gdy aplikacja działa w tle.



Pomyśl o `onSaveInstanceState()`wezwaniu jako o środku bezpieczeństwa; daje to szansę na zapisanie niewielkiej ilości informacji w pakiecie, gdy Twoja aktywność wyjdzie na pierwszy plan. System zapisuje teraz te dane, ponieważ jeśli zaczekał, aż zamknie Twoją aplikację, system może być pod presją zasobów.

Uwaga: czasami system Android wyłącza cały proces aplikacji, w tym wszystkie działania związane z aplikacją. Android wykonuje tego rodzaju zamknięcie, gdy system jest obciążony

i istnieje niebezpieczeństwo wizualnego opóźnienia, więc w tym momencie nie są uruchamiane żadne dodatkowe wywołania zwrotne ani kod. Proces Twojej aplikacji jest po prostu zamykany w tle. Ale dla użytkownika nie wygląda na to, że aplikacja została zamknięta. Gdy użytkownik nawiguje z powrotem do aplikacji, która została zamknięta przez system Android, system Android uruchamia tę aplikację ponownie. Będziesz chciał upewnić się, że użytkownik nie doświadczy żadnej utraty danych, gdy to się stanie.

Każdorazowe zapisanie danych zapewnia, że zaktualizowane dane w paczce są dostępne do odtworzenia, jeśli zajdzie taka potrzeba.

1. W MainActivityprogramie zastąp onSaveInstanceState()wywołanie zwrotne i dodaj instrukcję dziennika.

```
override fun onSaveInstanceState(outState: Bundle) {
  super.onSaveInstanceState(outState)

  Log.d(TAG, "onSaveInstanceState Called")
}
```

Uwaga: istnieją dwa nadpisania dla onSaveInstanceState(), jedno zawierające tylko outStateparametr, a drugie zawierające outStatei outPersistentStateparametry. Użyj tego pokazanego w powyższym kodzie, z jednym outStateparametrem.

2. Skompiluj i uruchom aplikację, a następnie kliknij przycisk **Strona główna** , aby umieścić ją w tle. Zauważ, że onSaveInstanceState()wywołanie zwrotne następuje zaraz po onPause()i onStop():

```
2020-10-16 11:05:21.726 23415-23415/com.example.android.dessertclicker D/MainActivity: onPause Called
2020-10-16 11:05:22.382 23415-23415/com.example.android.dessertclicker D/MainActivity: onStop Called
2020-10-16 11:05:22.393 23415-23415/com.example.android.dessertclicker D/MainActivity: onSaveInstanceState Called
```

3. Na górze pliku, tuż przed definicją klasy, dodaj te stałe:

```
const val KEY_REVENUE = "revenue_key"
const val KEY_DESSERT_SOLD = "dessert_sold_key"
```

Będziesz używać tych kluczy zarówno do zapisywania, jak i pobierania danych z pakietu stanów instancji.

4. Przewiń w dół do onSaveInstanceState()i zwróć uwagę na outStateparametr typu Bundle. A [Bundle]to zbiór par klucz-wartość, gdzie klucze są zawsze ciągami. W pakiecie można umieścić proste dane, takie jak Inti wartości. BooleanPonieważ system przechowuje ten pakiet w pamięci, najlepszym rozwiązaniem jest utrzymywanie małych danych w pakiecie. Rozmiar tego pakietu jest również ograniczony, chociaż rozmiar różni się w zależności od urządzenia. Jeśli przechowujesz zbyt dużo danych, ryzykujesz awarię aplikacji z powodu TransactionTooLargeExceptionbłędu.

5. W onSaveInstanceState(), umieść revenuewartość (liczbę całkowitą) w pakiecie za pomocą putInt()metody:

```
outState.putInt(KEY_REVENUE, revenue)
```

Metoda putInt()(i podobne metody z Bundleklasy jak putFloat()i putString()przyjmuje dwa argumenty: ciąg znaków dla klucza (KEY_REVENUEstała) i rzeczywistą wartość do zapisania.

6. Powtórz ten sam proces z liczbą sprzedanych deserów:

```
outState.putInt(KEY_DESSERT_SOLD, dessertsSold)
```

Użyj onCreate(), aby przywrócić dane pakietu

Stan aktywności można przywrócić w onCreate(Bundle)lub onRestoreInstanceState(Bundle)(Bundlewypełniona przez onSaveInstanceState()metoda zostanie przekazana do obu metod wywołania zwrotnego cyklu życia).

1. Przewiń w górę do onCreate()i sprawdź podpis metody:

```
override fun onCreate(savedInstanceState: Bundle?) {
```

Zauważ, że przy każdym wywołaniu onCreate()otrzymuje się a . BundleGdy aktywność zostanie wznowiona z powodu zamknięcia procesu, zapisany pakiet zostanie przekazany do onCreate(). Jeśli Twoja aktywność zaczynała się od nowa, Bundlejest onCreate()to null. Jeśli więc pakiet nie jest null, wiesz, że „odtwarzasz” działanie z wcześniej znanego punktu.

Uwaga: Jeśli aktywność jest ponownie tworzona, onRestoreInstanceState()wywołanie zwrotne jest wywoływane po onStart(), również z pakietem. W większości przypadków stan aktywności jest przywracany w onCreate(). Ale ponieważ onRestoreInstanceState()jest wywoływany po onStart(), jeśli kiedykolwiek będziesz musiał przywrócić jakiś stan po onCreate()wywołaniu, możesz użyć onRestoreInstanceState().

2. Dodaj ten kod do onCreate()tuż po ustawieniu bindingzmiennej:

```
if (savedInstanceState != null) {
   revenue = savedInstanceState.getInt(KEY_REVENUE, 0)
}
```

Test for nullokreśla, czy w pakiecie znajdują się dane, czy też pakiet to null, co z kolei informuje, czy aplikacja została uruchomiona od nowa, czy też została ponownie utworzona po zamknięciu. Ten test jest typowym wzorcem przywracania danych z pakietu.

Zauważ, że użyty tutaj klawisz (KEY_REVENUE) jest tym samym klawiszem, którego użyłeś putInt(). Aby mieć pewność, że za każdym razem używasz tego samego klucza, najlepiej jest zdefiniować te klucze jako stałe. Używasz getInt()go do pobierania danych z pakietu, tak jak putInt()w przypadku umieszczania danych w pakiecie. Metoda getInt()przyjmuje dwa argumenty:

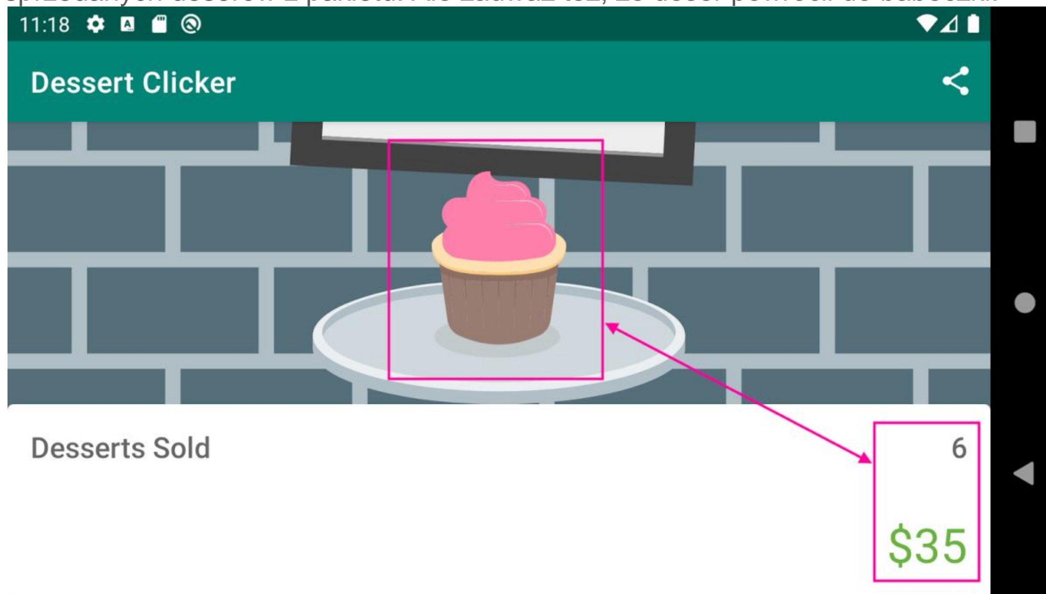
- Ciąg, który pełni rolę klucza, na przykład "key_revenue"dla wartości przychodu.
- Wartość domyślna na wypadek braku wartości dla tego klucza w pakiecie.

Liczba całkowita uzyskana z pakietu jest następnie przypisywana do revenuezmiennej, a interfejs użytkownika użyje tej wartości.

3. Dodaj getInt()metody przywracania przychodów i liczby sprzedanych deserów.

```
if (savedInstanceState != null) {
  revenue = savedInstanceState.getInt(KEY_REVENUE, 0)
  dessertsSold = savedInstanceState.getInt(KEY_DESSERT_SOLD, 0)
}
```

4. Skompiluj i uruchom aplikację. Naciśnij ciastko co najmniej pięć razy, aż zmieni się w pączek.
5. Obróć urządzenie. Zauważ, że tym razem aplikacja wyświetla prawidłowe wartości przychodów i sprzedanych deserów z pakietu. Ale zauważ też, że deser powrócił do babeczki.



Pozostała jeszcze jedna rzecz do zrobienia, aby upewnić się, że aplikacja powróci z zamknięcia dokładnie tak, jak została.

6. W `MainActivity`programie sprawdź `showCurrentDessert()`metodę. Zwróć uwagę, że ta metoda określa, który obraz deseru powinien być wyświetlany w działaniu, na podstawie aktualnej liczby sprzedanych deserów oraz listy deserów w `allDesserts`zmiennej.

```
for (dessert in allDesserts) {
  if (dessertsSold >= dessert.startProductionAmount) {
     newDessert = dessert
  }
   else break
}
```

Ta metoda polega na liczbie sprzedanych deserów, aby wybrać odpowiedni obraz. Dlatego nie musisz nic robić, aby przechowywać odwołanie do obrazu w pakiecie w programie `onSaveInstanceState()`. W tym zestawie już przechowujesz liczbę sprzedanych deserów.

7. W `onCreate()`bloku przywracającym stan z paczki wywołaj `showCurrentDessert()`:

```
if (savedInstanceState != null) {
  revenue = savedInstanceState.getInt(KEY_REVENUE, 0)
  dessertsSold = savedInstanceState.getInt(KEY_DESSERT_SOLD, 0)
  showCurrentDessert()
```

```
}
```

8. Skompiluj i uruchom aplikację oraz obróć ekran. Zwróć uwagę, że zarówno wartości sprzedanych deserów, całkowity przychód, jak i obraz deseru zostały poprawnie przywrócone.

6. Podsumowanie

Cykl życia aktywności

- Cykl *życia działania* to zestaw stanów, przez które migruje działanie. Cykl życia działania rozpoczyna się, gdy działanie jest tworzone po raz pierwszy i kończy się, gdy działanie zostanie zniszczone.
- Gdy użytkownik nawiguje między działaniami oraz wewnątrz i na zewnątrz aplikacji, każde działanie przechodzi między stanami w cyklu życia działania.
- Każdy stan w cyklu życia aktywności ma odpowiadającą mu metodę wywołania zwrotnego, którą można nadpisać w swojej `Activity`klasie. Podstawowy zestaw metod cyklu życia to:`onCreate()onStart()onPause()onRestart()onResume()onStop()onDestroy()`
- Aby dodać zachowanie, które występuje, gdy aktywność przechodzi w stan cyklu życia, zastąp metodę wywołania zwrotnego stanu.
- Aby dodać metody zastępowania szkieletu do swoich klas w Android Studio, wybierz opcję **Code > Override Methods** lub naciśnij `Control+o`.

Logowanie za pomocą dziennika

- Interfejs API rejestrowania systemu Android, a w szczególności `Log`klasa, umożliwia pisanie krótkich wiadomości, które są wyświetlane w Logcat w Android Studio.
- Służy `Log.d()`do pisania wiadomości debugowania. *Ta metoda przyjmuje dwa argumenty: tag* dziennika (zazwyczaj nazwa klasy) oraz *komunikat* dziennika (krótki ciąg).
- Użyj okna **Logcat** w Android Studio, aby wyświetlić dzienniki systemowe, w tym wiadomości, które piszesz.

Zachowywanie stanu aktywności

- Gdy aplikacja działa w tle, zaraz po `onStop()`wywołaniu, dane aplikacji można zapisać w pakiecie. Niektóre dane aplikacji, takie jak zawartość `EditText`, są automatycznie zapisywane.
- Pakiet jest instancją `Bundle`, która jest kolekcją kluczy i wartości. Klucze są zawsze strunami.
- Użyj `onSaveInstanceState()`wywołania zwrotnego, aby zapisać inne dane w pakiecie, który chcesz zachować, nawet jeśli aplikacja została automatycznie zamknięta. Aby umieścić dane w pakiecie, użyj metod pakietu rozpoczynających się od `put`, takich jak `putInt()`.
- Możesz odzyskać dane z pakietu w `onRestoreInstanceState()`metodzie lub częściej w `onCreate()`. Metoda `onCreate()`ma `savedInstanceState`parametr, który przechowuje pakiet.
- Jeśli `savedInstanceState`zmienna to `null`, działanie zostało uruchomione bez pakietu stanów i nie ma danych stanu do pobrania.
- Aby pobrać dane z pakietu z kluczem, użyj `Bundle`metod zaczynających się od `get`, takich jak `getInt()`.

Zmiany w konfiguracji

- Zmiana *konfiguracji* ma miejsce, gdy stan urządzenia zmienia się tak radykalnie, że najłatwiejszym sposobem rozwiązania tej zmiany przez system jest zniszczenie i odbudowanie aktywności.
- Najczęstszym przykładem zmiany konfiguracji jest zmiana przez użytkownika urządzenia z trybu pionowego na poziomy lub z trybu poziomego na pionowy. Zmiana konfiguracji może również nastąpić w przypadku zmiany języka urządzenia lub podłączenia klawiatury sprzętowej.
- Gdy nastąpi zmiana konfiguracji, system Android wywołuje wszystkie wywołania zwrotne zamknięcia cyklu życia aktywności. Następnie system Android ponownie uruchamia działanie od zera, uruchamiając wszystkie wywołania zwrotne podczas uruchamiania cyklu życia.
- Gdy system Android zamyka aplikację z powodu zmiany konfiguracji, ponownie uruchamia działanie z pakietem stanu, który jest dostępny dla `onCreate()`.
- Podobnie jak w przypadku zamykania procesu, zapisz stan aplikacji w pakiecie w `onSaveInstanceState()`.

7. Dowiedz się więcej

- `Log`klasa
- Zapisuj i przeglądaj logi za pomocą Logcat
- Stałe czasu kompilacji
- `Activity`klasa
- `AppCompatActivity`klasa
- Przewodnik dla programistów aktywności

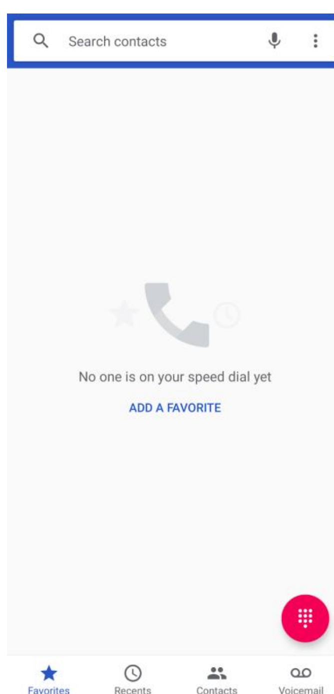
Wprowadzenie do komponentu Nawigacja

Dowiedz się o składniku architektury nawigacji w systemie Android Jetpack, który zapewnia strukturę do tworzenia nawigacji w aplikacji.

Fragmenty i komponent nawigacyjny

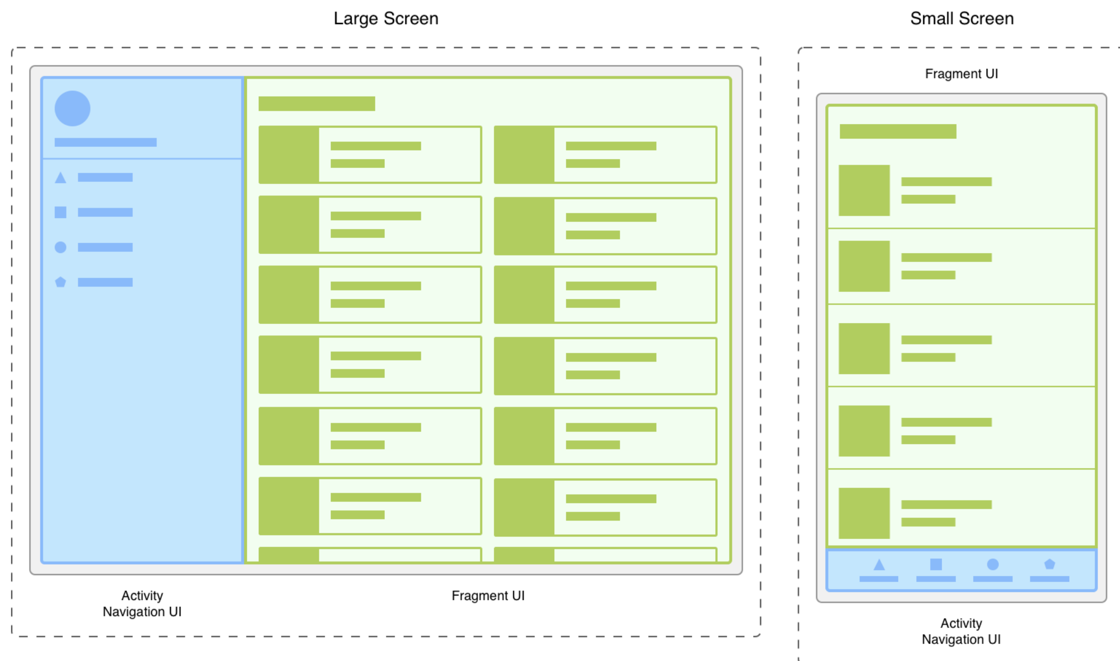
1. Zanim zaczniesz

W laboratorium programowania Działania i intencje dodano intencje w aplikacji Words , aby nawigować między dwoma działaniami. Chociaż jest to przydatny wzorzec nawigacji, który warto znać, to tylko część historii tworzenia dynamicznych interfejsów użytkownika dla aplikacji. Wiele aplikacji na Androida nie wymaga osobnej aktywności na każdym ekranie. W rzeczywistości wiele typowych wzorców interfejsu użytkownika, takich jak zakładki, istnieje w ramach jednej czynności, używając czegoś, co nazywa się *fragmentami* .



Fragment to element interfejsu użytkownika wielokrotnego użytku ; fragmenty mogą być ponownie wykorzystane i osadzone w jednym lub kilku działaniach. Na powyższym zrzucie ekranu stuknięcie w kartę nie wyzwala zamiaru wyświetlenia następnego ekranu. Zamiast tego przełączanie kart po prostu zamienia poprzedni fragment na inny. Wszystko to dzieje się bez uruchamiania kolejnej aktywności.

Możesz nawet wyświetlać wiele fragmentów jednocześnie na jednym ekranie, na przykład układ główny-szczegółowy dla tabletów. W poniższym przykładzie zarówno interfejs nawigacji po lewej, jak i zawartość po prawej stronie mogą być zawarte w osobnym fragmencie. Oba fragmenty istnieją jednocześnie w tej samej czynności.



Jak widać, fragmenty są integralną częścią budowania wysokiej jakości aplikacji. Podczas tego ćwiczenia z programowania nauczysz się podstaw fragmentów i przekonwertujesz aplikację Words, aby ich używać. Dowiesz się również, jak korzystać z komponentu Jetpack Navigation i pracować z nowym plikiem zasobów o nazwie **Navigation Graph** , aby nawigować między fragmentami w tej samej aktywności hosta. Pod koniec tego ćwiczenia z programowania zdobędziesz podstawowe umiejętności wdrażania fragmentów w następnej aplikacji.

Warunki wstępne

Przed ukończeniem tego ćwiczenia z programowania powinieneś wiedzieć

- Jak dodać zasoby XML i pliki Kotlin do projektu Android Studio.
- Jak działa cykl życia aktywności na wysokim poziomie.
- Jak zastąpić i zaimplementować metody w istniejącej klasie.
- Jak tworzyć instancje klas Kotlin, uzyskiwać dostęp do właściwości klas i wywoływać metody.
- Podstawowa znajomość wartości dopuszczających i nie dopuszczających wartości null oraz wiedza, jak bezpiecznie obsługiwać wartości null.

Czego się nauczysz

- Jak cykl życia fragmentu różni się od cyklu życia działania.
- Jak przekonwertować istniejącą aktywność na fragment.
- Jak dodawać miejsca docelowe do wykresu nawigacyjnego i przekazywać dane między fragmentami podczas korzystania z wtyczki Safe Args.

Co zbudujesz

- Zmodyfikujesz aplikację Words tak, aby używała jednego działania i wielu fragmentów oraz poruszała się między fragmentami za pomocą składnika nawigacyjnego.

Czego potrzebujesz

- Komputer z zainstalowanym Android Studio.
- Kod rozwiązania aplikacji Words z laboratorium kodowania Działania i zamiary

2. Kod startowy

W tym ćwiczeniu z programowania dowiesz się, w którym miejscu zostało przerwane, w aplikacji Words na końcu ćwiczenia z programowania Działania i Intencje. Jeśli ukończyłeś już ćwiczenia z programowania dotyczące działań i zamiarów, możesz użyć swojego kodu jako punktu wyjścia. Możesz alternatywnie pobrać kod do tego momentu z GitHub.

Pobierz kod startowy do tego ćwiczenia z programowania

To ćwiczenie z programowania zapewnia kod startowy, który można rozszerzyć o funkcje nauczane w tym ćwiczeniu z programowania. Kod startowy może zawierać kod, który jest Ci znany z poprzednich ćwiczeń z programowania. Może również zawierać kod, który jest Tobie nieznany i o którym dowiesz się podczas późniejszych ćwiczeń z programowania.

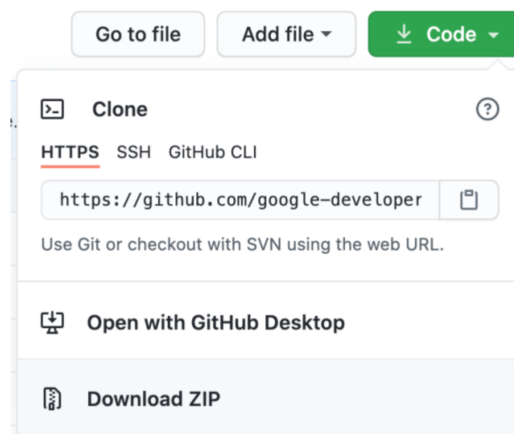
Jeśli używasz kodu startowego z GitHub, pamiętaj, że nazwa folderu to `android-basics-kotlin-words-app-activities`. Wybierz ten folder podczas otwierania projektu w Android Studio.

URL kodu startowego: https://github.com/google-developer-training/android-basics-kotlin-words-app/tree/activities

Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

Pobierz kod

1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.

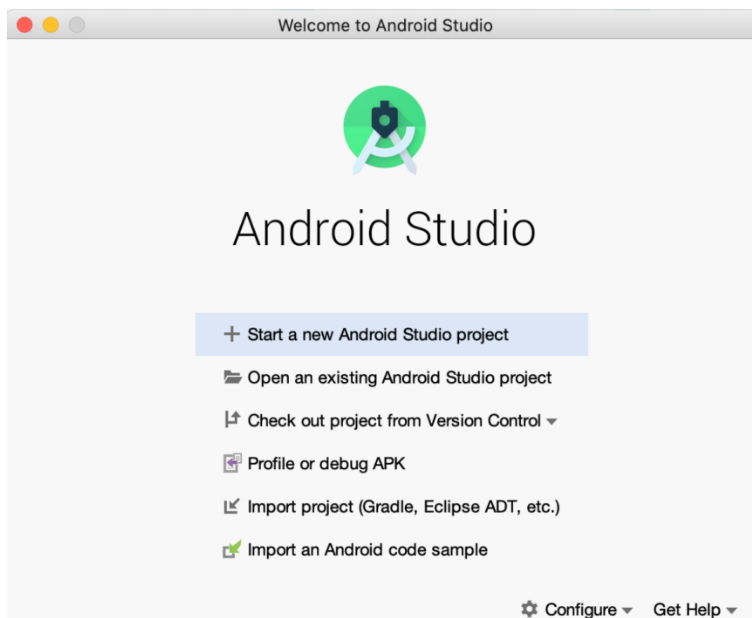


3. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
4. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**

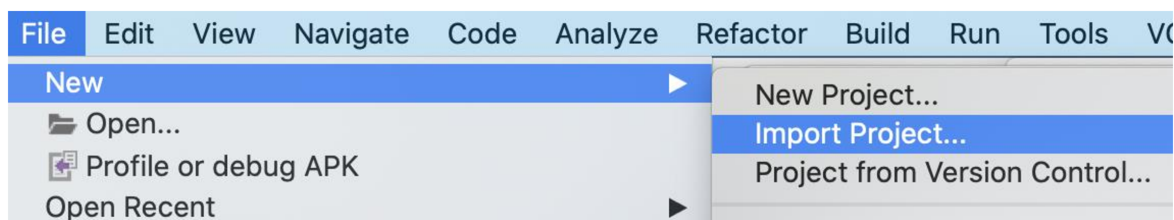
5. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

Otwórz projekt w Android Studio

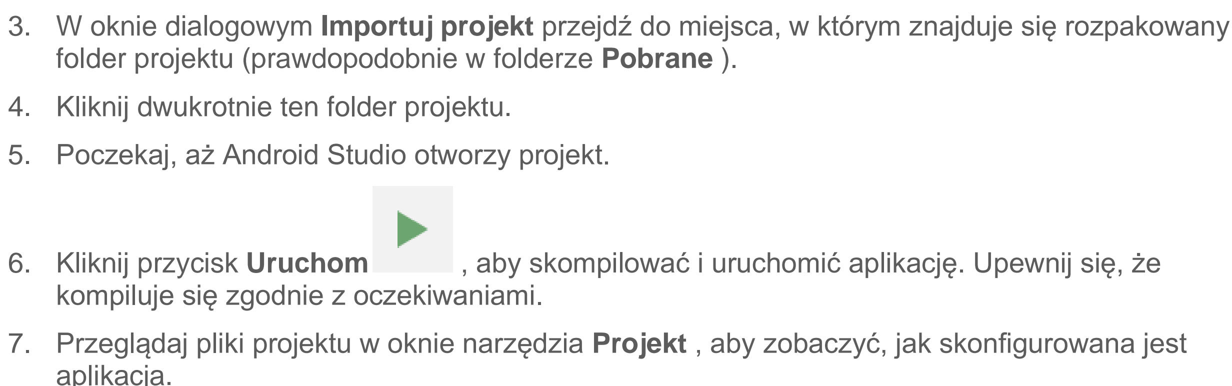
1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz istniejący projekt Android Studio.**



Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Nowy** > **Importuj projekt** .



3. W oknie dialogowym **Importuj projekt** przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.



6. Kliknij przycisk **Uruchom** , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.
7. Przeglądaj pliki projektu w oknie narzędzia **Projekt** , aby zobaczyć, jak skonfigurowana jest aplikacja.

3. Fragmenty i cykl życia fragmentów

Fragment to po prostu element interfejsu użytkownika aplikacji, który można ponownie wykorzystać. Podobnie jak działania, fragmenty mają cykl życia i mogą reagować na dane wejściowe użytkownika. Fragment jest zawsze zawarty w hierarchii widoków działania, gdy jest wyświetlany na ekranie. Ze względu na nacisk na ponowne użycie i modułowość, możliwe jest nawet jednoczesne hostowanie wielu fragmentów w ramach jednej czynności. Każdy fragment zarządza odrębnym cyklem życia.

Cykl życia fragmentu

Podobnie jak czynności, fragmenty mogą być inicjowane i usuwane z pamięci, a przez całe swoje istnienie pojawiać się, znikać i pojawiać ponownie na ekranie. Podobnie jak działania, fragmenty mają cykl życia z kilkoma stanami i zapewniają kilka metod, które można przesłonić, aby odpowiadać na przejścia między nimi. Cykl życia fragmentu ma pięć stanów reprezentowanych przez wyliczenie *Lifecycle.State .*

- ZAINICJALIZOWANO: Utworzono nową instancję fragmentu.
- UTWORZONE: Wywoływane są pierwsze metody cyklu życia fragmentów. W tym stanie tworzony jest również widok skojarzony z fragmentem.
- ROZPOCZĘTO: Fragment jest widoczny na ekranie, ale nie ma „fokusa”, co oznacza, że nie może odpowiadać na dane wejściowe użytkownika.
- WZNOWIONE: Fragment jest widoczny i ma skupienie.
- DESTROYED: Obiekt fragmentu został usunięty z instancji.

Podobnie jak w przypadku działań, `Fragment`klasa udostępnia wiele metod, które można przesłonić, aby odpowiadać na zdarzenia cyklu życia.

- `onCreate()`: Fragment został utworzony i jest w `CREATED`stanie. Jednak odpowiedni widok nie został jeszcze utworzony.
- `onCreateView()`: Ta metoda polega na nadmuchaniu układu. Fragment wszedł w `CREATED`stan.
- `onViewCreated()`: Ta funkcja jest wywoływana po utworzeniu widoku. W tej metodzie zazwyczaj powiążesz określone widoki z właściwościami, wywołując `findViewById()`.
- `onStart()`: Fragment wszedł w `STARTED`stan.
- `onResume()`: Fragment wszedł w `RESUMED`stan i ma teraz fokus (może odpowiadać na dane wprowadzone przez użytkownika).
- `onPause()`: Fragment ponownie wszedł w `STARTED`stan. Interfejs użytkownika jest widoczny dla użytkownika
- `onStop()`: Fragment ponownie wszedł w `CREATED`stan. Obiekt jest tworzony, ale nie jest już wyświetlany na ekranie.
- `onDestroyView()`: Wywoływane tuż przed wejściem fragmentu w `DESTROYED`stan. Widok został już usunięty z pamięci, ale obiekt fragmentu nadal istnieje.
- `onDestroy()`: Fragment wchodzi w `DESTROYED`stan.

Poniższy wykres podsumowuje cykl życia fragmentów i przejścia między stanami.

Lifecycle State	Callback
CREATED	onCreate()
	onCreateView()
	onViewCreated()
STARTED	onStart()
RESUMED	onResume()
STARTED	onPause()
CREATED	onStop()
	onDestroyView()
DESTROYED	onDestroy()

Stany cyklu życia i metody wywołań zwrotnych są bardzo podobne do tych używanych w przypadku działań. Należy jednak pamiętać o różnicy w `onCreate()`metodzie. W przypadku działań użyjesz tej metody do rozdmuchania układu i powiązania widoków. Jednak w cyklu życia fragmentu `onCreate()`jest wywoływana przed utworzeniem widoku, więc nie można tutaj zawyżać układu. Zamiast tego robisz to w `onCreateView()`. Następnie, po utworzeniu widoku, `onViewCreated()`wywoływana jest metoda, w której można następnie powiązać właściwości z określonymi widokami.

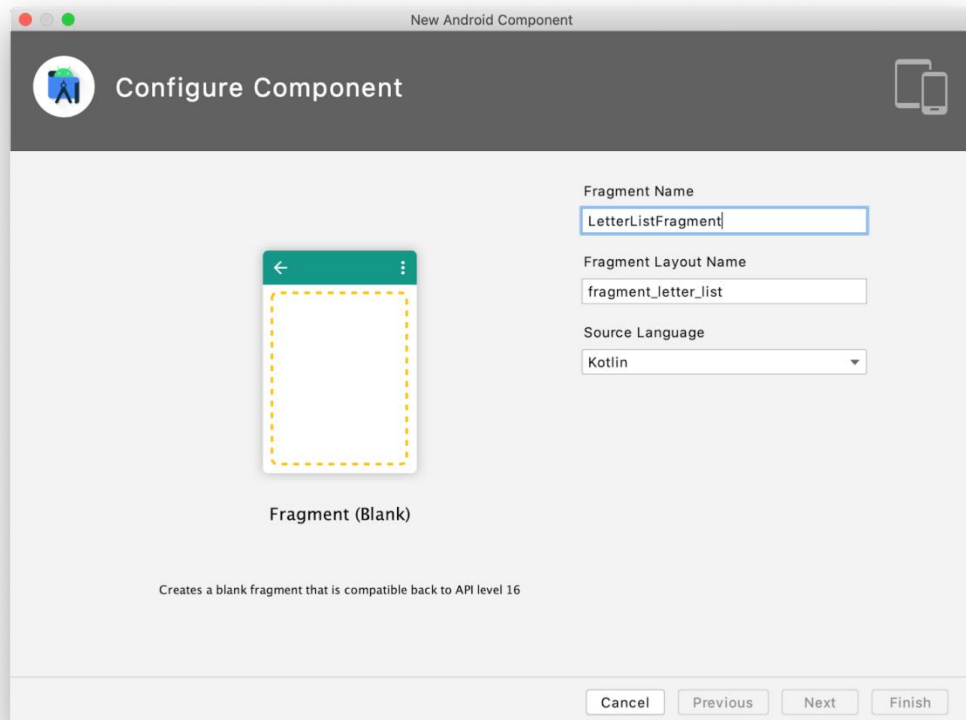
Choć prawdopodobnie brzmiało to jak dużo teorii, teraz znasz podstawy działania fragmentów oraz ich podobieństwa i różnic w stosunku do czynności. W pozostałej części tego ćwiczenia z programowania wykorzystasz tę wiedzę. Najpierw przeniesiesz aplikację Words, nad którą pracowałeś wcześniej, do układu opartego na fragmentach. Następnie zaimplementujesz nawigację między fragmentami w ramach jednego działania.

4. Utwórz pliki fragmentów i układów

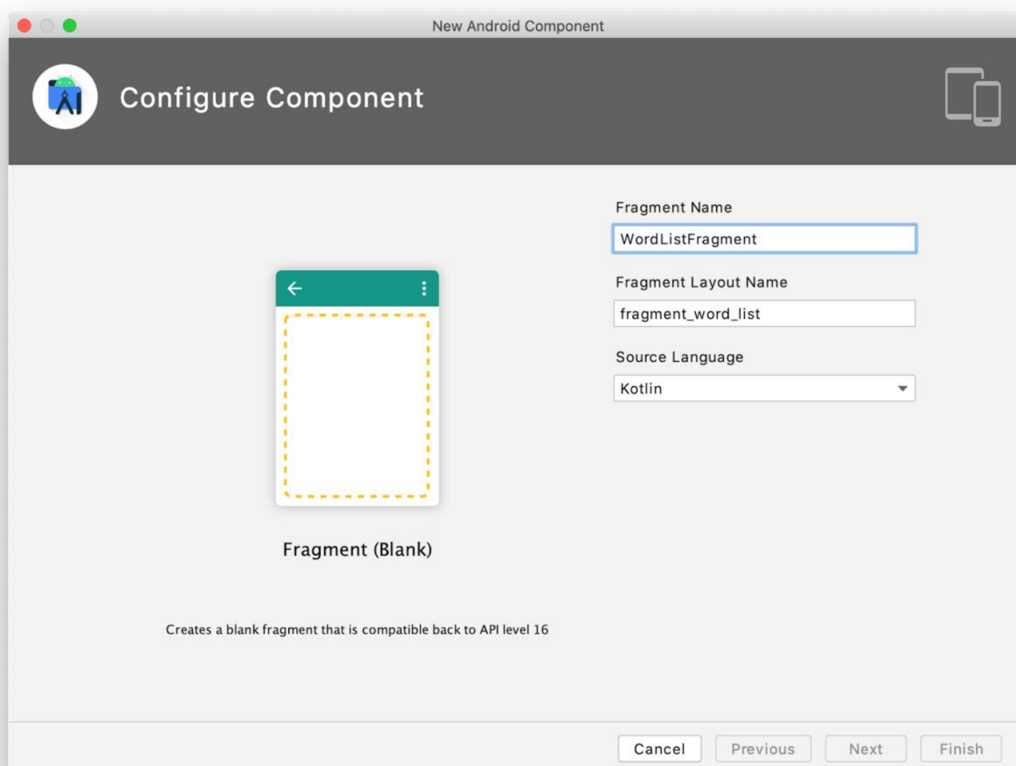
Podobnie jak w przypadku działań, każdy dodany fragment będzie składał się z dwóch plików — pliku XML dla układu i klasy Kotlin do wyświetlania danych i obsługi interakcji użytkownika. Dodasz fragment zarówno do listy liter, jak i listy słów.

1. Po wybraniu **aplikacji** w Nawigatorze projektu dodaj następujące fragmenty (**Plik > Nowy > Fragment > Fragment (Puste)**) i dla każdego z nich należy wygenerować plik klasy i układu.

- Dla pierwszego fragmentu ustaw nazwę **fragmentu** na `LetterListFragment`. Nazwa **układu fragmentu** powinna być wypełniona jako `fragment_letter_list`.



- Dla drugiego fragmentu ustaw nazwę **fragmentu** na `WordListFragment`. Nazwa **układu fragmentu** powinna być wypełniona jako `fragment_word_list.xml`.



2. Wygenerowane klasy Kotlin dla obu fragmentów zawierają dużo kodu standardowego, powszechnie używanego podczas implementacji fragmentów. Jednak, ponieważ uczysz się o fragmentach po raz pierwszy, śmiało usuń wszystko poza deklaracją klasy dla LetterListFragmenti WordListFragmentz obu plików. Przeprowadzimy Cię przez implementację fragmentów od podstaw, abyś wiedział, jak działa cały kod. Po usunięciu kodu wzorcowego pliki Kotlina powinny wyglądać następująco.

   ListListFragment.kt

```
package com.example.wordsapp

import androidx.fragment.app.Fragment

class LetterListFragment : Fragment() {

}
```

   WordListFragment.kt

```
package com.example.wordsapp

import androidx.fragment.app.Fragment

class WordListFragment : Fragment() {

}
```

3. Skopiuj zawartość activity_main.xmldo fragment_letter_list.xmli zawartość activity_detail.xmldo fragment_word_list.xml. Zaktualizuj tools:contextdo i do . fragment_letter_list.xml_ _.LetterListFragmenttools:contextfragment_word_list.xml.WordListFragment

   Po zmianach pliki układu fragmentów powinny wyglądać następująco.

   fragment_listy_lista.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".LetterListFragment">

   <androidx.recyclerview.widget.RecyclerView
       android:id="@+id/recycler_view"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:clipToPadding="false"
       android:padding="16dp" />
```

```
</FrameLayout>
```

fragment_word_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".WordListFragment">

   <androidx.recyclerview.widget.RecyclerView
       android:id="@+id/recycler_view"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:clipToPadding="false"
       android:padding="16dp"
       tools:listitem="@layout/item_view" />

</FrameLayout>
```

5. Implementuj fragment listy listów

Podobnie jak w przypadku działań, musisz nadmuchać układ i powiązać poszczególne widoki. Istnieje tylko kilka drobnych różnic podczas pracy z cyklem życia fragmentów. Przeprowadzimy Cię przez proces konfiguracji `LetterListFragment`, a następnie będziesz mieć możliwość zrobienia tego samego w przypadku `WordListFragment`.

Aby zaimplementować powiązanie widoku w programie `LetterListFragment`, musisz najpierw uzyskać odwołanie dopuszczające wartość null do programu `FragmentLetterListBinding`. Klasy powiązań, takie jak ta, są generowane przez Android Studio dla każdego pliku układu, gdy `viewBinding`właściwość jest włączona w `buildFeatures`sekcji pliku **build.gradle .** Musisz tylko przypisać właściwości w swojej klasie fragmentu dla każdego widoku w `FragmentLetterListBinding`.

Typ powinien być `FragmentLetterListBinding?`i powinien mieć początkową wartość `null`. Dlaczego warto ją unieważnić? Ponieważ nie można nadmuchać układu, dopóki nie `onCreateView()`zostanie wywołany. Istnieje pewien okres czasu pomiędzy utworzeniem instancji `LetterListFragment`(kiedy jej cykl życia zaczyna się od `onCreate()`) a momentem, w którym ta właściwość jest rzeczywiście użyteczna. Pamiętaj również, że widoki fragmentów mogą być tworzone i niszczone kilka razy w całym cyklu życia fragmentu. Z tego powodu należy również zresetować wartość w innej metodzie cyklu życia, `onDestroyView()`.

1. W `LetterListFragment.kt`programie zacznij od uzyskania odwołania do `FragmentLetterListBinding`i nazwij odwołanie `_binding`.

   ```
   private var _binding: FragmentLetterListBinding? = null
   ```

Ponieważ dopuszczalna jest wartość null, za każdym razem, gdy uzyskujesz dostęp do właściwości _binding, (np . _binding?.someView), musisz uwzględnić ?for null safety. Nie oznacza to jednak, że musisz zaśmiecać swój kod znakami zapytania tylko z powodu jednej wartości null. Jeśli masz pewność, że wartość nie będzie null, gdy uzyskasz do niej dostęp, możesz dołączyć !!do jej nazwy typu. Następnie możesz uzyskać do niego dostęp jak do każdej innej właściwości, bez ?operatora.

UWAGA: Podczas tworzenia zmiennej null przy użyciu !!, dobrym pomysłem jest ograniczenie jej użycia tylko do jednego lub kilku miejsc, w których wiadomo, że wartość nie będzie null, tak jak wiesz, że _bindingbędzie miała wartość po jej przypisaniu w onCreateView(). Dostęp do wartości null w ten sposób jest niebezpieczny i może prowadzić do awarii, więc używaj go oszczędnie, jeśli w ogóle.

2. Utwórz nową właściwość o nazwie bindowanie (bez podkreślenia) i ustaw ją na _binding!!.

```
private val binding get() = _binding!!
```

Tutaj get()oznacza, że ta właściwość jest „tylko do pobrania”. Oznacza to, że możesz **uzyskać** wartość, ale po przypisaniu (tak jak tutaj) nie możesz przypisać jej do niczego innego.

UWAGA: W Kotlinie i ogólnie w programowaniu często napotkasz nazwy właściwości poprzedzone podkreśleniem. Zwykle oznacza to, że właściwość nie jest przeznaczona do bezpośredniego dostępu. W Twoim przypadku uzyskujesz dostęp do powiązania widoku LetterListFragmentz właściwością powiązania. Nie _bindingma jednak konieczności uzyskiwania dostępu do nieruchomości poza LetterListFragment.

3. Aby wyświetlić menu opcji, zastąp przekazywanie połączenia onCreate().wewnętrznego .onCreate()setHasOptionsMenu()true

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setHasOptionsMenu(true)
}
```

4. Pamiętaj, że przy fragmentach układ jest zawyżony w onCreateView(). Zaimplementuj onCreateView(), zwiększając widok, ustawiając wartość _bindingi zwracając widok główny.

```
override fun onCreateView(
  inflater: LayoutInflater, container: ViewGroup?,
  savedInstanceState: Bundle?
): View? {
  _binding = FragmentLetterListBinding.inflate(inflater, container, false)
  val view = binding.root
  return view
}
```

5. Poniżej bindingwłaściwości utwórz właściwość dla widoku recyklera.

```
private lateinit var recyclerView: RecyclerView
```

6. Następnie ustaw wartość recyclerViewwłaściwości w onViewCreated()i wywołaj chooseLayout()tak jak w MainActivity. Wkrótce przeniesiesz tę chooseLayout()metodę LetterListFragment, więc nie martw się, że wystąpił błąd.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
  recyclerView = binding.recyclerView
  chooseLayout()
}
```

   Zwróć uwagę, jak klasa powiązania już utworzyła właściwość dla recyclerViewi nie musisz wywoływać findViewById()dla każdego widoku.

7. Na koniec w onDestroyView()programie zresetuj _bindingwłaściwość na null, ponieważ widok już nie istnieje.

```
override fun onDestroyView() {
  super.onDestroyView()
  _binding = null
}
```

8. Jedyną inną rzeczą, na którą należy zwrócić uwagę, jest to, że istnieją pewne subtelne różnice w onCreateOptionsMenu()metodzie podczas pracy z fragmentami. Chociaż Activityklasa ma właściwość globalną o nazwie menuInflater, Fragment nie ma tej właściwości. Inflater menu jest zamiast tego przekazywany do onCreateOptionsMenu(). Zauważ też, że onCreateOptionsMenu()metoda używana z fragmentami nie wymaga instrukcji return. Zaimplementuj metodę, jak pokazano:

```
override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
  inflater.inflate(R.menu.layout_menu, menu)

  val layoutButton = menu.findItem(R.id.action_switch_layout)
  setIcon(layoutButton)
}
```

9. Przenieś pozostały kod dla chooseLayout(), setIcon()i onOptionsItemSelected()ze stanu, MainActivityw jakim jest. Jedyną inną różnicą, na którą należy zwrócić uwagę, jest to, że w przeciwieństwie do czynności, fragment nie jest [Context](). Nie możesz przekazać this(odnosząc się do obiektu fragmentu) jako kontekst menedżera układu. Jednak fragmenty zapewniają contextwłaściwość, której można użyć zamiast tego. Reszta kodu jest identyczna z MainActivity.

```
private fun chooseLayout() {
  when (isLinearLayoutManager) {
    true -> {
      recyclerView.layoutManager = LinearLayoutManager(context)
      recyclerView.adapter = LetterAdapter()
    }
    false -> {
      recyclerView.layoutManager = GridLayoutManager(context, 4)
      recyclerView.adapter = LetterAdapter()
    }
  }
}
```

```
private fun setIcon(menuItem: MenuItem?) {
  if (menuItem == null)
      return

  menuItem.icon =
      if (isLinearLayoutManager)
          ContextCompat.getDrawable(this.requireContext(), R.drawable.ic_grid_layout)
      else ContextCompat.getDrawable(this.requireContext(), R.drawable.ic_linear_layout)
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
  return when (item.itemId) {
      R.id.action_switch_layout -> {
          isLinearLayoutManager = !isLinearLayoutManager
          chooseLayout()
          setIcon(item)

          return true
      }

      else -> super.onOptionsItemSelected(item)
  }
}
```

Uwaga : requireContext()zwraca Contextten fragment, z którym jest aktualnie powiązany.

10. Na koniec skopiuj isLinearLayoutManagerwłaściwość z MainActivity. Umieść to tuż pod deklaracją recyclerViewwłasności.

```
private var isLinearLayoutManager = true
```

11. Teraz, gdy cała funkcjonalność została przeniesiona do LetterListFragment, jedyne, co MainActivityklasa musi zrobić, to nadmuchać układ, aby fragment był wyświetlany w widoku. Śmiało i usuń wszystko onCreate()oprócz MainActivity. Po zmianach MainActivitypowinien zawierać tylko poniższe.

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)

  val binding = ActivityMainBinding.inflate(layoutInflater)
  setContentView(binding.root)
}
```

Twoja kolej

To tyle, jeśli chodzi o migrację MainActivitydo LettersListFragment. Migracja DetailActivityjest prawie identyczna. Wykonaj poniższe czynności, aby przeprowadzić migrację kodu do WordListFragment.

1. Skopiuj obiekt towarzyszący z DetailActivitydo WordListFragment. Upewnij się, że odwołanie do SEARCH_PREFIXin WordAdapterzostało zaktualizowane do odwołania WordListFragment.
2. Dodaj _bindingzmienną. Zmienna powinna mieć dopuszczalną wartość null i mieć nulljako wartość początkową.
3. Dodaj zmienną typu „dobierz tylko” o nazwie bindowanie równej _bindingzmiennej.
4. Napełnij układ w onCreateView()programie , ustawiając wartość _bindingi zwracając widok główny.
5. Wykonaj wszystkie pozostałe ustawienia w onViewCreated(): uzyskaj odniesienie do widoku recyklera, ustaw jego menedżera układu i adapter oraz dodaj dekorację elementu. Musisz zdobyć list z intencji. Ponieważ fragmenty nie mają intentwłaściwości i zwykle nie powinny mieć dostępu do intencji aktywności nadrzędnej. Na razie odwołujesz się do activity.intent(zamiast) intent, DetailActivityaby uzyskać dodatki.
6. Zresetuj _bindingdo wartości null w onDestroyView.
7. Usuń pozostały kod z DetailActivity, pozostawiając tylko onCreate()metodę.

   Spróbuj samodzielnie przejść wszystkie kroki, zanim przejdziesz dalej. Szczegółowy opis przejścia jest dostępny w następnym kroku.

6. Konwertuj DetailActivity na WordListFragment

Mamy nadzieję, że migracja DetailActivitydo WordListFragment. Jest to prawie identyczne z migracją MainActivitydo programu LetterListFragment. Jeśli utkniesz w dowolnym momencie, kroki są podsumowane poniżej.

1. Najpierw skopiuj obiekt towarzyszący do WordListFragment.

```
companion object {
  val LETTER = "letter"
  val SEARCH_PREFIX = "https://www.google.com/search?q="
}
```

2. Następnie w miejscu LetterAdapter, w onClickListener()którym realizujesz zamiar, musisz zaktualizować wywołanie na , putExtra()zastępując .DetailActivity.LETTERWordListFragment.LETTER

```
intent.putExtra(WordListFragment.LETTER, holder.button.text.toString())
```

3. Podobnie WordAdaptermusisz zaktualizować miejsce , w onClickListener()którym przechodzisz do wyników wyszukiwania słowa, DetailActivity.SEARCH_PREFIXzastępując WordListFragment.SEARCH_PREFIX.

```
val queryUrl: Uri = Uri.parse("${WordListFragment.SEARCH_PREFIX}${item}")
```

4. Wracając WordListFragment, dodajesz zmienną wiążącą typu FragmentWordListBinding?.

```
private var _binding: FragmentWordListBinding? = null
```

5. Następnie tworzysz zmienną typu „tylko do pobrania”, dzięki czemu możesz odwoływać się do widoków bez konieczności używania ?.

```
private val binding get() = _binding!!
```

6. Następnie zawyżasz układ, przypisując `_binding`zmienną i zwracając widok główny. Pamiętaj, że dla fragmentów robisz to w `onCreateView()`, a nie `onCreate()`.

```
override fun onCreateView(
  inflater: LayoutInflater,
  container: ViewGroup?,
  savedInstanceState: Bundle?
): View? {
  _binding = FragmentWordListBinding.inflate(inflater, container, false)
  return binding.root
}
```

7. Następnie wdrażasz `onViewCreated()`. Jest to prawie identyczne z konfiguracją wejścia `recyclerView`w `onCreate()`programie `DetailActivity`. Jednak ponieważ fragmenty nie mają bezpośredniego dostępu do `intent`, musisz odwoływać się do niego za pomocą `activity.intent`. Musisz to `onViewCreated()`jednak zrobić, ponieważ nie ma gwarancji, że aktywność będzie istniała wcześniej w cyklu życia.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
  val recyclerView = binding.recyclerView
  recyclerView.layoutManager = LinearLayoutManager(requireContext())
  recyclerView.adapter = WordAdapter(activity?.intent?.extras?.getString(LETTER).toString(),
requireContext())

  recyclerView.addItemDecoration(
     DividerItemDecoration(context, DividerItemDecoration.VERTICAL)
  )
}
```

8. Na koniec możesz zresetować `_binding`zmienną w `onDestroyView()`.

```
override fun onDestroyView() {
  super.onDestroyView()
  _binding = null
}
```

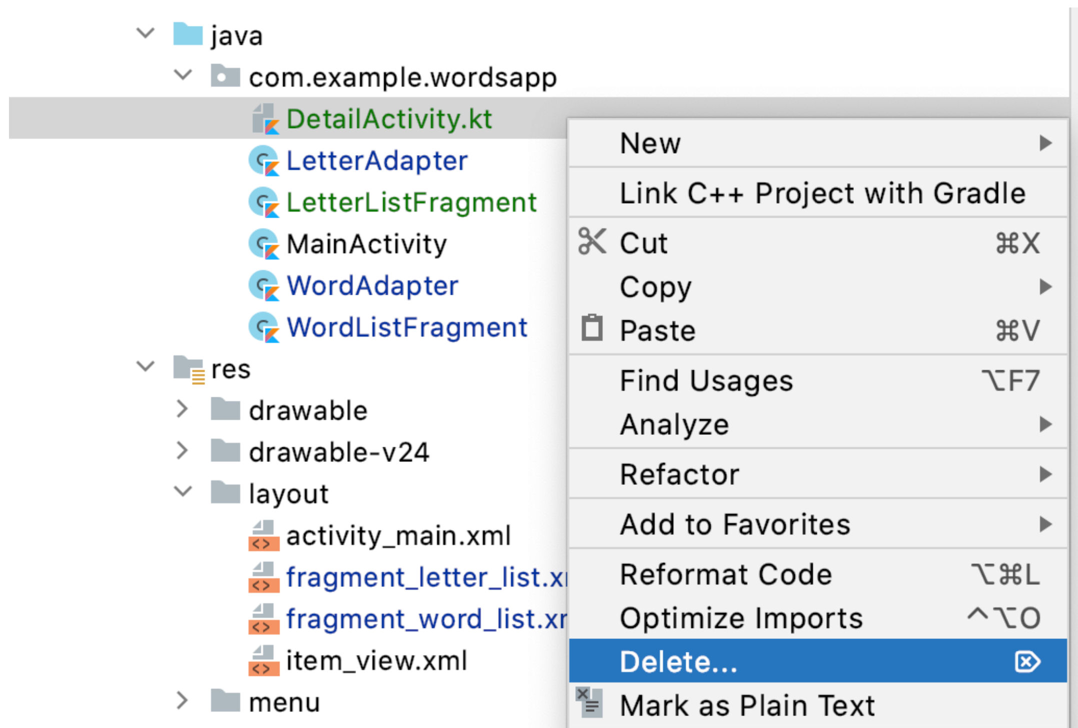
9. Po przeniesieniu całej tej funkcjonalności do WordListFragment, możesz teraz usunąć kod z DetailActivity. Pozostaje tylko metoda onCreate().

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)

  val binding = ActivityDetailBinding.inflate(layoutInflater)
  setContentView(binding.root)
}
```

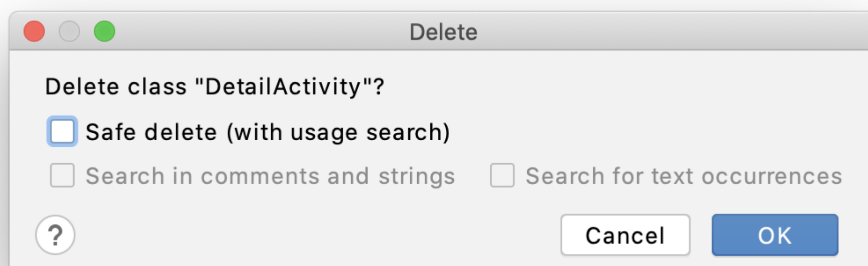
Usuń aktywność szczegółów

Teraz, po pomyślnej migracji funkcji programu DetailActivitydo programu WordListFragment, nie potrzebujesz już DetailActivity. Możesz śmiało usunąć zarówno DetailActivity.kti activity_detail.xml, jak i wprowadzić niewielką zmianę w manifeście.

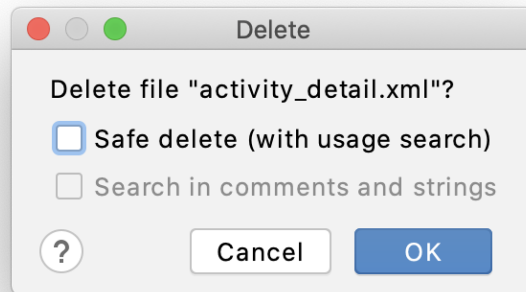
1. Najpierw usuńDetailActivity.kt



2. Upewnij się, że **Bezpieczne usuwanie** nie jest zaznaczone i kliknij **OK** .



3. Następnie usuń activity_detail.xml. Ponownie upewnij się, że **Bezpieczne usuwanie** nie jest zaznaczone.



4. Wreszcie, ponieważ DetailActivityjuż nie istnieje, usuń następujące z AndroidManifest.xml.

```
<activity
  android:name=".DetailActivity"
  android:parentActivityName=".MainActivity" />
```

Po usunięciu działania szczegółów pozostają dwa fragmenty (LetterListFragment i WordListFragment) oraz jedno działanie (MainActivity). W następnej sekcji dowiesz się o komponencie Jetpack Navigation i edytuj activity_main.xmlgo, aby mógł wyświetlać i nawigować między fragmentami, zamiast hostować układ statyczny.

7. Komponent nawigacji Jetpack

Android Jetpack udostępnia *komponent Nawigacja,* który pomoże Ci obsłużyć dowolną implementację nawigacji, prostą lub złożoną, w Twojej aplikacji. Komponent Nawigacja składa się z trzech kluczowych części, których będziesz używać do implementacji nawigacji w aplikacji **Words** .

- Wykres nawigacji: wykres nawigacji to plik XML, który zapewnia wizualną reprezentację nawigacji w Twojej aplikacji. Plik składa się z *miejsc docelowych* , które odpowiadają poszczególnym czynnościom i fragmentom, a także czynności między nimi, które można wykorzystać w kodzie do nawigacji z jednego miejsca do drugiego. Podobnie jak w przypadku plików układu, Android Studio zapewnia wizualny edytor do dodawania miejsc docelowych i działań do wykresu nawigacji.
- NavHost: A NavHostsłuży do wyświetlania miejsc docelowych z wykresu nawigacji w ramach aktywności. Podczas nawigowania między fragmentami miejsce docelowe pokazane w NavHostjest aktualizowane. Użyjesz wbudowanej implementacji o nazwie NavHostFragment, w swoim MainActivity.
- NavController: NavControllerObiekt umożliwia sterowanie nawigacją pomiędzy celami wyświetlanymi w NavHost. Podczas pracy z intencjami trzeba było wywołać startActivity, aby przejść do nowego ekranu. Za pomocą komponentu Nawigacja możesz wywołać metodę NavController, navigate()aby zamienić wyświetlany fragment. Pomaga również NavControllerw wykonywaniu typowych zadań, takich jak reagowanie na systemowy przycisk „w górę”, aby przejść z powrotem do poprzednio wyświetlanego fragmentu.

Zależność nawigacji

1. build.gradleW pliku na poziomie projektu w **buildscript > ext** poniżej material_versionustaw nav_versionrówną 2.3.1.

```
buildscript {
   ext {
      appcompat_version = "1.2.0"
      constraintlayout_version = "2.0.2"
      core_ktx_version = "1.3.2"
      kotlin_version = "1.3.72"
      material_version = "1.2.1"
      nav_version = "2.3.1"
   }

   ...


}
```

2. W pliku na poziomie build.gradleaplikacji dodaj następujące elementy do grupy zależności.

```
implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"
implementation "androidx.navigation:navigation-ui-ktx:$nav_version"
```

Bezpieczna wtyczka Args

Kiedy po raz pierwszy zaimplementowałeś nawigację w aplikacji **Words** , użyłeś wyraźnej intencji między tymi dwoma działaniami. Aby przekazać dane między tymi dwoma działaniami, wywołałeś putExtra()metodę, przekazując wybraną literę.

Zanim zaczniesz implementować komponent Nawigacja w aplikacji **Words** , dodasz także coś, co nazywa **się Safe Args** — wtyczkę Gradle, która pomoże Ci zapewnić bezpieczeństwo podczas przekazywania danych między fragmentami.

Wykonaj następujące kroki, aby zintegrować SafeArgs z projektem.

1. W pliku najwyższego poziomu build.gradlew **buildscript > dependencies** dodaj następującą ścieżkę klasy.

```
classpath "androidx.navigation:navigation-safe-args-gradle-plugin:$nav_version"
```

2. build.gradleW pliku na poziomie aplikacji, pluginsu góry, dodaj androidx.navigation.safeargs.kotlin.

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

3. Po wyedytowaniu plików Gradle u góry może pojawić się żółty baner z prośbą o zsynchronizowanie projektu. Kliknij „ **Synchronizuj teraz** ” i poczekaj minutę lub dwie, aż Gradle zaktualizuje zależności Twojego projektu, aby odzwierciedlić zmiany.

Po zakończeniu synchronizacji możesz przejść do następnego kroku, w którym dodasz wykres nawigacyjny.

8. Korzystanie z wykresu nawigacyjnego

Teraz, gdy masz podstawową wiedzę na temat fragmentów i ich cyklu życia, nadszedł czas, aby sprawy stały się nieco ciekawsze. Następnym krokiem jest włączenie komponentu Nawigacja. Komponent *nawigacji* po prostu odnosi się do zbioru narzędzi do implementacji nawigacji, w szczególności między fragmentami. Będziesz pracować z nowym edytorem wizualnym, który pomoże zaimplementować nawigację między fragmentami; Wykres nawigacji (lub w skrócie NavGraph).

Co to jest wykres nawigacyjny?

Wykres nawigacji (lub w skrócie NavGraph) to wirtualne mapowanie nawigacji Twojej aplikacji. Każdy ekran lub fragment w twoim przypadku staje się możliwym „celem”, do którego można nawigować. A `NavGraph`może być reprezentowane przez plik XML pokazujący, w jaki sposób poszczególne miejsca docelowe odnoszą się do siebie.

Za kulisami to faktycznie tworzy nową instancję `NavGraph`klasy. Jednak miejsca docelowe z wykresu nawigacyjnego są wyświetlane użytkownikowi przez `FragmentContainerView`. Wszystko, co musisz zrobić, to utworzyć plik XML i zdefiniować możliwe miejsca docelowe. Następnie możesz użyć wygenerowanego kodu do poruszania się między fragmentami.

Użyj FragmentContainerView w MainActivity

Ponieważ Twoje układy są teraz zawarte w `fragment_letter_list.xml`i `fragment_word_list.xml`, `activity_main.xml`plik nie musi już zawierać układu pierwszego ekranu w aplikacji. Zamiast tego zmienisz przeznaczenie, `MainActivity`aby zawierał a, `FragmentContainerView`aby działał jako NavHost dla twoich fragmentów. Od tego momentu cała nawigacja w aplikacji będzie odbywać się w ramach `FragmentContainerView`.

1. Zastąp zawartość pliku `FrameLayout`w **activity_main.xml** , który jest `androidx.recyclerview.widget.RecyclerView`znakiem `FragmentContainerView`. Nadaj mu identyfikator `nav_host_fragment`i ustaw jego wysokość i szerokość tak, `match_parent`aby wypełniały cały układ ramki.

   Zastąp to:

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recycler_view"
    ...
    android:padding="16dp" />
```

Z tym:

```
<androidx.fragment.app.FragmentContainerView
   android:id="@+id/nav_host_fragment"
   android:layout_width="match_parent"
   android:layout_height="match_parent" />
```

2. Poniżej atrybutu id dodaj `name`atrybut i ustaw go na `androidx.navigation.fragment.NavHostFragment`. Chociaż możesz określić konkretny fragment dla tego atrybutu, ustawienie go na `NavHostFragment`umożliwia `FragmentContainerView`nawigację między fragmentami.

```
android:name="androidx.navigation.fragment.NavHostFragment"
```

3. Poniżej atrybutów layout_height i layout_width dodaj wywołany atrybut `app:defaultNavHost`i ustaw go na `"true"`. Pozwala to kontenerowi fragmentów na interakcję z hierarchią nawigacji. Na przykład, jeśli zostanie naciśnięty systemowy przycisk wstecz, kontener przejdzie z powrotem do poprzednio pokazanego fragmentu, tak jak to się dzieje, gdy prezentowana jest nowa czynność.

```
app:defaultNavHost="true"
```

4. Dodaj wywoływany atrybut `app:navGraph`i ustaw go na `"@navigation/nav_graph"`. Wskazuje to na plik XML, który definiuje, w jaki sposób fragmenty Twojej aplikacji mogą nawigować do siebie. Na razie studio Android pokaże nierozwiązany błąd symbolu. Zajmiesz się tym w następnym zadaniu.

```
app:navGraph="@navigation/nav_graph"
```

5. Wreszcie, ponieważ dodałeś dwa atrybuty z przestrzenią nazw aplikacji, pamiętaj o dodaniu atrybutu xmlns:app do `FrameLayout`.

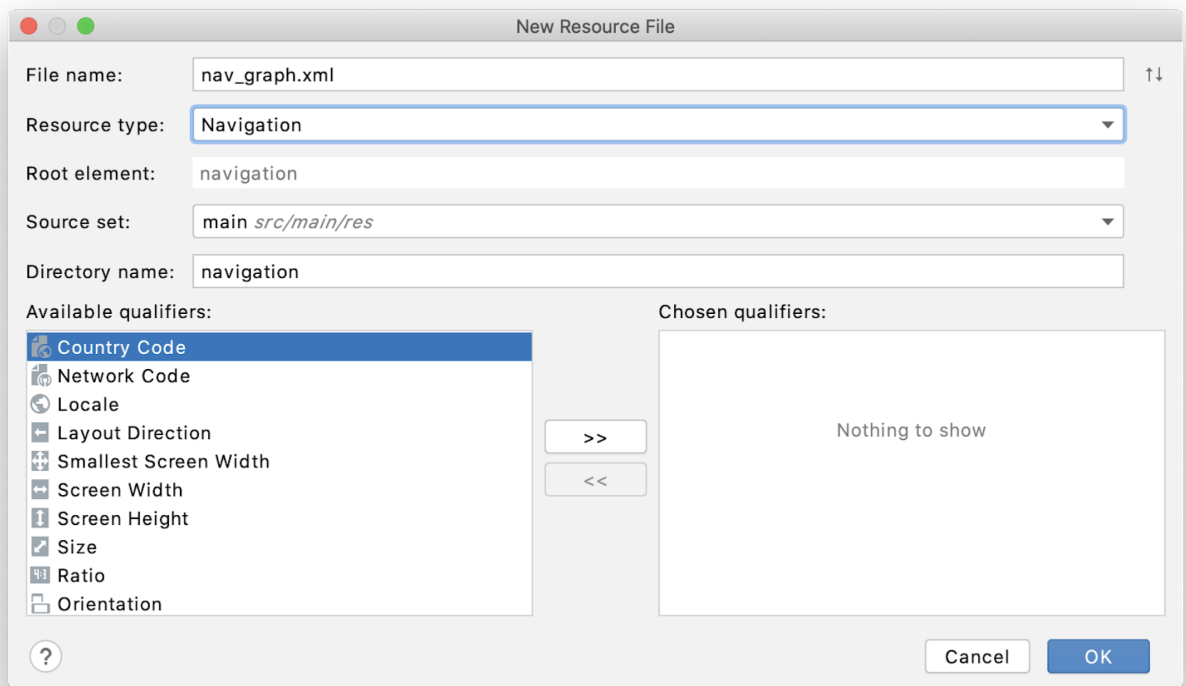
```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">
```

To wszystkie zmiany w activity_main.xml. Następnie utworzysz `nav_graph`plik.

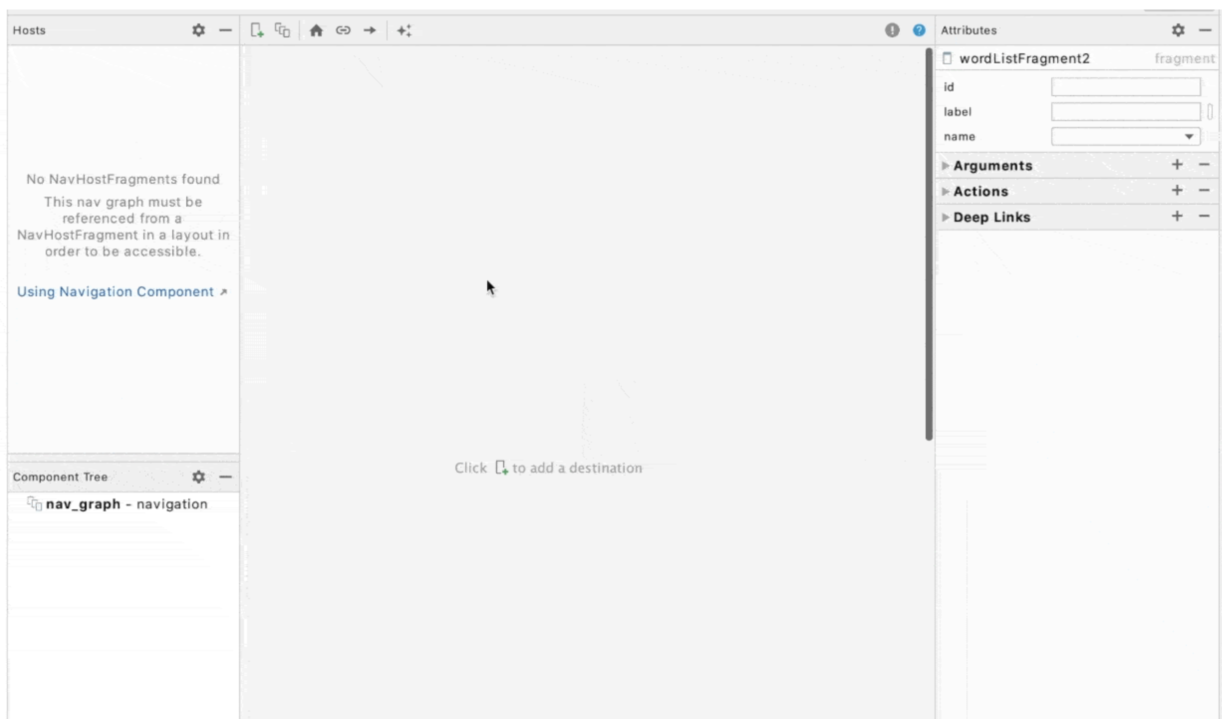
Skonfiguruj wykres nawigacji

Dodaj plik wykresu nawigacji (**Plik > Nowy > Plik zasobów systemu Android**) i wypełnij pola w następujący sposób.

- Nazwa pliku: `nav_graph.xml`.jest taka sama, jak nazwa ustawiona dla `app:navGraph`atrybutu.
- Typ zasobu: **Nawigacja** . Nazwa **katalogu** powinna wtedy automatycznie zmienić się na nawigację. Zostanie utworzony nowy folder zasobów o nazwie „nawigacja”.



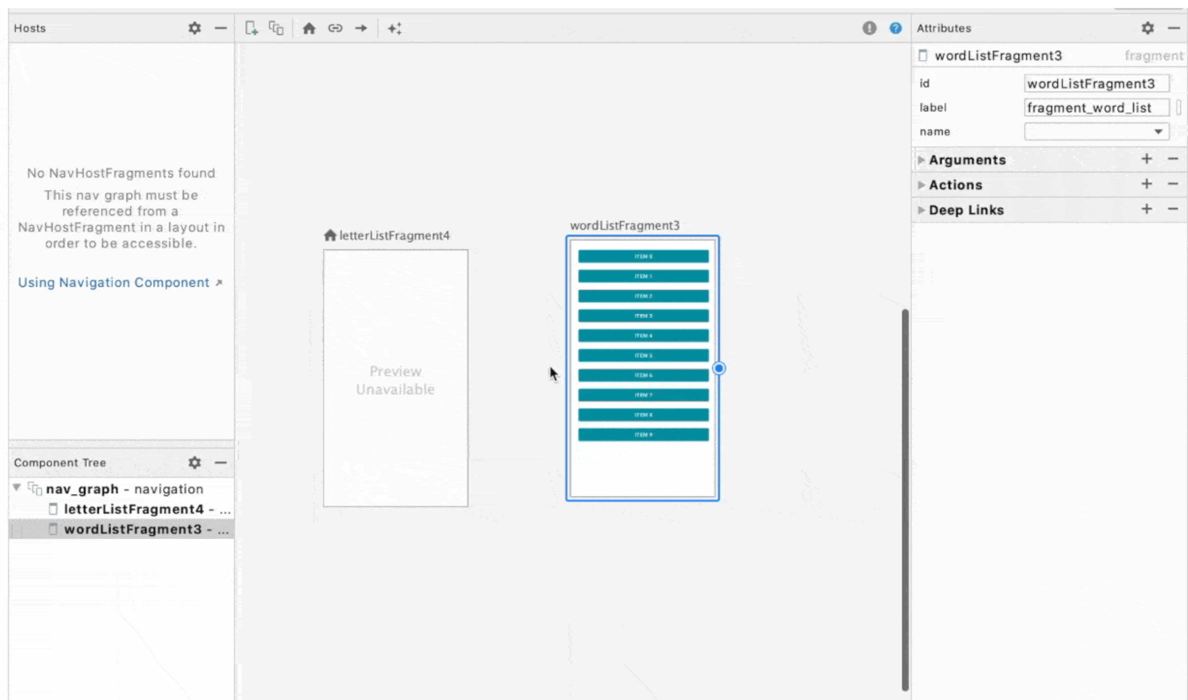
Po utworzeniu pliku XML otrzymujesz nowy edytor wizualny. Ponieważ już odwołałeś się nav_graphdo właściwości , aby dodać nowe miejsce docelowe, kliknij nowy przycisk w lewym górnym rogu ekranu i utwórz miejsce docelowe dla każdego fragmentu (jeden dla i jeden dla).FragmentContainerViewnavGraphfragment_letter_listfragment_word_list



Po dodaniu fragmenty te powinny pojawić się na wykresie nawigacyjnym na środku ekranu. Możesz także wybrać określone miejsce docelowe za pomocą drzewa komponentów, które pojawi się po lewej stronie.

Utwórz akcję nawigacji

Aby utworzyć akcję nawigacyjną między letterListFragmentmiejscami wordListFragmentdocelowymi, najedź myszą na miejsce letterListFragmentdocelowe i przeciągnij z okręgu, który pojawi się po prawej stronie, do wordListFragmentmiejsca docelowego.



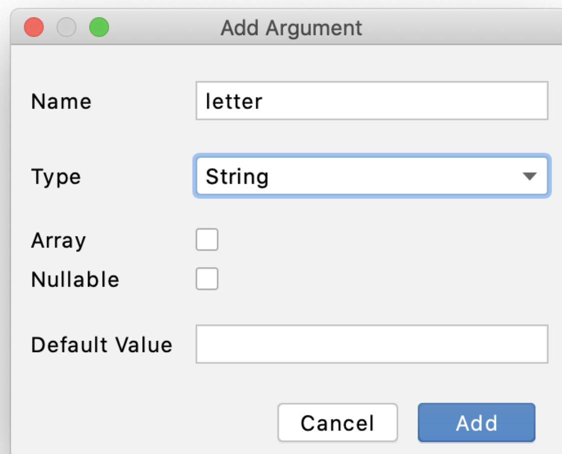
Powinieneś teraz zobaczyć strzałkę, która została utworzona, aby reprezentować akcję między dwoma miejscami docelowymi. Kliknij strzałkę, a zobaczysz w okienku atrybutów, że ta akcja ma nazwę , do action_letterListFragment_to_wordListFragmentktórej można się odwoływać w kodzie.

Określ argumenty dla WordListFragment

Podczas nawigowania między działaniami przy użyciu intencji określiłeś „dodatkowe", aby wybrana litera mogła zostać przekazana do wordListFragment. Nawigacja obsługuje również przekazywanie parametrów między miejscami docelowymi, a plus robi to w sposób bezpieczny dla typu.

Wybierz miejsce wordListFragmentdocelowe i w okienku atrybutów w obszarze **Argumenty** kliknij przycisk plus, aby utworzyć nowy argument.

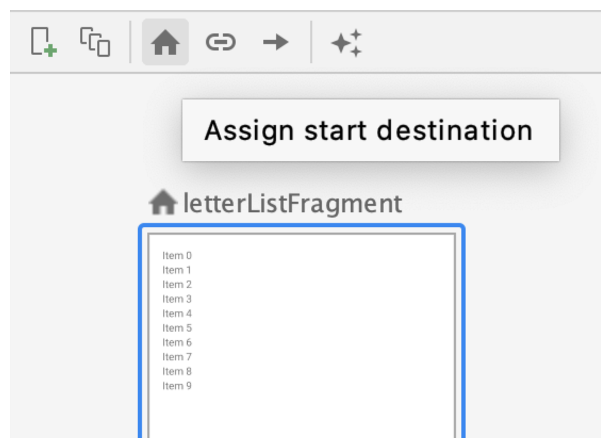
Argument powinien być wywołany letter, a typ powinien być String. W tym miejscu pojawia się dodana wcześniej wtyczka Safe Args. Określenie tego argumentu jako ciągu zapewnia, Stringże podczas wykonywania akcji nawigacyjnej w kodzie będzie oczekiwany a.



Ustawianie początkowego miejsca docelowego

Podczas gdy NavGraph jest świadomy wszystkich potrzebnych miejsc docelowych, skąd będzie FragmentContainerViewwiedział, który fragment ma zostać wyświetlony jako pierwszy? W NavGraph musisz ustawić listę liter jako miejsce docelowe.

Ustaw miejsce docelowe, wybierając letterListFragmenti klikając przycisk **Przypisz miejsce docelowe początkowe** .



1. To wszystko, co musisz teraz zrobić z edytorem NavGraph. W tym momencie śmiało zbuduj projekt. W Android Studio wybierz **Build > Rebuild Project** z paska menu. Spowoduje to wygenerowanie kodu na podstawie wykresu nawigacji, dzięki czemu będziesz mógł użyć właśnie utworzonej akcji nawigacji.

Wykonaj akcję nawigacji

Otwórz LetterAdapter. ktaby wykonać akcję nawigacji. Wymaga to tylko dwóch kroków.

1. Usuń zawartość przycisku setOnClickListener(). Zamiast tego musisz pobrać właśnie utworzoną akcję nawigacyjną. Dodaj następujące elementy do setOnClickListener().

```
val action = LetterListFragmentDirections.actionLetterListFragmentToWordListFragment(letter =
holder.button.text.toString())
```

Prawdopodobnie nie rozpoznajesz niektórych z tych nazw klas i funkcji, a to dlatego, że zostały one automatycznie wygenerowane po zbudowaniu projektu. W tym miejscu pojawia się wtyczka Safe Args, którą dodałeś w pierwszym kroku — akcje utworzone w NavGraph są przekształcane w kod, którego możesz użyć. Nazwy jednak powinny być dość intuicyjne. `LetterListFragmentDirections`pozwala odnieść się do wszystkich możliwych ścieżek nawigacji, począwszy od `letterListFragment`. Funkcja`actionLetterListFragmentToWordListFragment()`

jest konkretną czynnością, aby przejść do`wordListFragment`.

Gdy masz odniesienie do akcji nawigacji, po prostu uzyskaj odwołanie do *kontrolera NavController* (obiektu, który umożliwia wykonywanie akcji nawigacji) i wywołaj `navigate()`przekazywanie w akcji.

```
holder.view.findNavController().navigate(action)
```

Skonfiguruj główną aktywność

Ostatni element konfiguracji to `MainActivity`. Wystarczy kilka zmian, `MainActivity`aby wszystko działało.

1. Utwórz `navController`właściwość. Jest to oznaczone jako `lateinit`, ponieważ zostanie ustawione w `onCreate`.

```
private lateinit var navController: NavController
```

2. Następnie, po wywołaniu `setContentView()`in `onCreate()`, uzyskaj odwołanie do `nav_host_fragment`(jest to identyfikator Twojego `FragmentContainerView`) i przypisz go do swojej `navController`właściwości.

```
val navHostFragment = supportFragmentManager
    .findFragmentById(R.id.nav_host_fragment) as NavHostFragment
navController = navHostFragment.navController
```

3. Następnie w `onCreate()`, zadzwoń `setupActionBarWithNavController()`, przechodząc `navController`. Zapewnia `LetterListFragment`to widoczność przycisków paska akcji (paska aplikacji), takich jak opcja menu.

```
setupActionBarWithNavController(navController)
```

4. Na koniec zaimplementuj `onSupportNavigateUp()`. Wraz z ustawieniem `defaultNavHost`w `true`XML, ta metoda umożliwia obsługę przycisku w **górę** . Jednak Twoja działalność musi zapewnić wdrożenie.

```
override fun onSupportNavigateUp(): Boolean {
  return navController.navigateUp() || super.onSupportNavigateUp()
}
```

W tym momencie wszystkie komponenty są na miejscu, aby nawigacja działała z fragmentami. Jednak teraz, gdy nawigacja jest wykonywana przy użyciu fragmentów zamiast intencji, dodatkowa intencja dla litery, której używasz WordListFragment, nie będzie już działać. W następnym kroku zaktualizujesz WordListFragment, aby uzyskać letterargument.

UWAGA: Ponieważ navigateUp()funkcja może się nie powieść, zwraca a określającą, Booleanczy się powiedzie. Musisz jednak zadzwonić tylko wtedy, super.onSupportNavigateUp()gdy navigateUp()wróci false. Działa to, ponieważ ||operator wymaga, aby był spełniony tylko jeden z warunków, więc jeśli navigateUp()zwraca true, prawa strona ||wyrażenia nigdy nie jest wykonywana. Jeśli natomiast navigateUp()ma wartość false, to wywoływana jest implementacja w klasie nadrzędnej. Nazywa się to oceną zwarcia i jest fajną małą sztuczką programistyczną, o której warto wiedzieć.

9. Pobieranie argumentów w WordListFragment

Wcześniej odwoływałeś activity?.intentsię , WordListFragmentaby uzyskać dostęp do letterdodatku. Chociaż to działa, nie jest to najlepsza praktyka, ponieważ fragmenty można osadzać w innych układach i w większej aplikacji, znacznie trudniej jest założyć, do której aktywności należy fragment. Co więcej, gdy nawigacja jest wykonywana przy użyciu nav_graphbezpiecznych argumentów, nie ma żadnych intencji, więc próba uzyskania dostępu do dodatkowych intencji po prostu nie zadziała.

Na szczęście dostęp do bezpiecznych argumentów jest dość prosty i nie musisz też czekać na onViewCreated()wywołanie.

1. W WordListFragmentprogramie utwórz letterIdwłaściwość. Możesz oznaczyć to jako lateinit, aby nie trzeba było ustawiać go na wartość null.

```
private lateinit var letterId: String
```

2. Następnie zastąp onCreate()(nie onCreateView()lub onViewCreated()!), dodaj następujące.

```
override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)


   arguments?.let {
       letterId = it.getString(LETTER).toString()
   }
}
```

Ponieważ może to argumentsbyć opcjonalne, zauważ, że wywołujesz let()i przekazujesz lambdę. Ten kod zostanie wykonany przy założeniu , że argumentsnie jest null, przekazując argumenty inne niż null dla itparametru. Jeśli jednak argumentsjest null, lambda nie zostanie wykonana.

```
arguments?.let { it: Bundle
    letterId = it.getString(LETTER).toString()
}
```

Chociaż nie jest to część rzeczywistego kodu, Android Studio zapewnia pomocną wskazówkę, aby uświadomić sobie itparametr.

Czym dokładnie jest Bundle? Pomyśl o tym jako o parze klucz-wartość używanej do przekazywania danych między klasami, takich jak działania i fragmenty. Właściwie korzystałeś już z pakietu, gdy zadzwoniłeś intent?.extras?.getString()podczas wykonywania zamiaru w pierwszej wersji tej aplikacji. Pobieranie łańcucha z argumentów podczas pracy z fragmentami działa dokładnie w ten sam sposób.

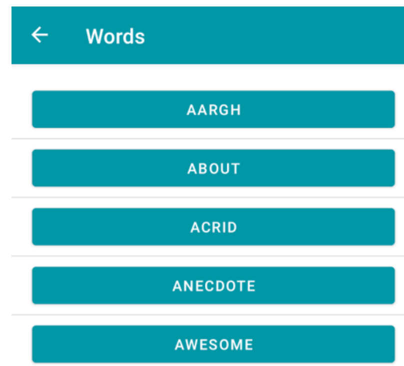
3. Wreszcie, możesz uzyskać dostęp letterIdpo ustawieniu adaptera widoku recyklera. Zastąp activity?.intent?.extras?.getString(LETTER).toString()na . onViewCreated()_letterId

```
recyclerView.adapter = WordAdapter(letterId, requireContext())
```

Zrobiłeś to! Poświęć chwilę na uruchomienie swojej aplikacji. Teraz jest w stanie poruszać się między dwoma ekranami, bez żadnych intencji, a wszystko to w ramach jednej czynności.

10. Zaktualizuj etykiety fragmentów

Udało Ci się przekonwertować oba ekrany na fragmenty. Przed wprowadzeniem jakichkolwiek zmian pasek aplikacji dla każdego fragmentu miał opisowy tytuł dla każdej aktywności zawartej na pasku aplikacji. Jednak po przekonwertowaniu na używane fragmenty brakuje tego tytułu w działaniu szczegółów.



Fragmenty mają właściwość o nazwie "label", w której można ustawić tytuł, o którym aktywność nadrzędna będzie wiedziała na pasku aplikacji.

1. W strings.xml, po nazwie aplikacji, dodaj następującą stałą.

   <string name="word_list_fragment_label">Words That Start With {letter}</string>

2. Możesz ustawić etykietę dla każdego fragmentu na wykresie nawigacyjnym. Wróć nav_graph.xmli wybierz letterListFragmentw drzewie komponentów, a w panelu atrybutów ustaw etykietę na app_nameciąg

letterListFragment	fragment
id	letterListFragment
label	@string/app_name
name	
▶ Arguments	+ −

3. Wybierz wordListFragmenti ustaw etykietę naword_list_fragment_label

wordListFragment	fragment
id	wordListFragment
label	@string/word_list_fragment_label
name	
▼ Arguments	+ −

Gratulacje, że dotarłeś tak daleko! Uruchom swoją aplikację jeszcze raz i powinieneś zobaczyć wszystko tak, jak na początku ćwiczenia z kodowania, tylko teraz cała Twoja nawigacja jest hostowana w jednym działaniu z osobnym fragmentem dla każdego ekranu.

11. Kod rozwiązania

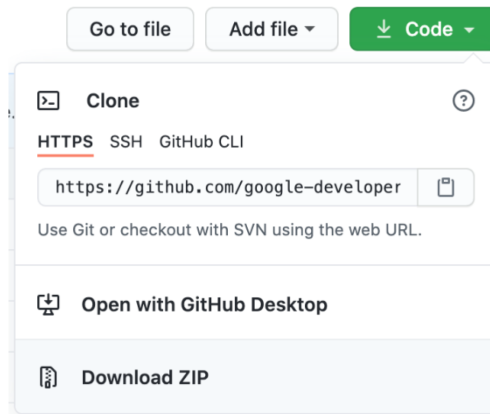
Kod rozwiązania dla tego ćwiczenia programowania znajduje się w projekcie pokazanym poniżej.

Adres URL kodu rozwiązania: https://github.com/google-developer-training/android-basics-kotlin-words-app/tree/main

Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

Pobierz kod

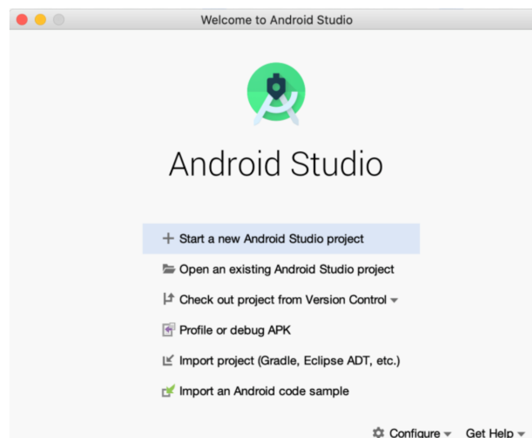
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



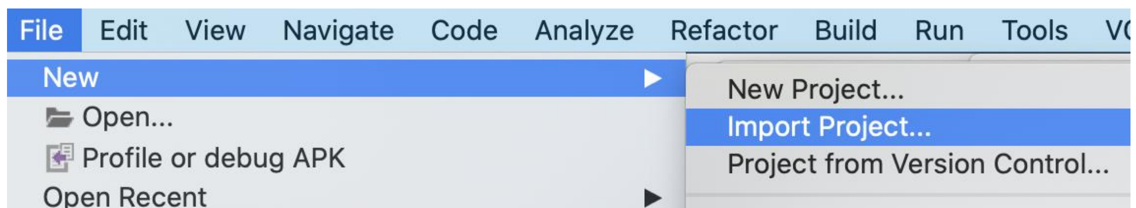
3. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
4. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
5. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

Otwórz projekt w Android Studio

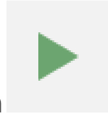
1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz istniejący projekt Android Studio.**



Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Nowy** > **Importuj projekt** .



3. W oknie dialogowym **Importuj projekt** przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.

6. Kliknij przycisk **Uruchom**  , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.
7. Przeglądaj pliki projektu w oknie narzędzia **Projekt** , aby zobaczyć, jak skonfigurowana jest aplikacja.

12. Podsumowanie

- Fragmenty to elementy interfejsu użytkownika wielokrotnego użytku, które można osadzić w działaniach.
- Cykl życia fragmentu różni się od cyklu życia działania, a konfiguracja widoku występuje w onViewCreated(), a nie w onCreateView().
- A FragmentContainerViewsłuży do osadzania fragmentów w innych działaniach i może zarządzać nawigacją między fragmentami.

Korzystanie z komponentu nawigacyjnego

- Ustawienie navGraphatrybutu a FragmentContainerViewpozwala na poruszanie się między fragmentami w ramach działania.
- Edytor NavGraphumożliwia dodawanie działań nawigacyjnych i określanie argumentów między różnymi miejscami docelowymi.
- Podczas gdy nawigacja przy użyciu intencji wymaga przekazania dodatków, składnik Nawigacja używa SafeArgs do automatycznego generowania klas i metod dla akcji nawigacji, zapewniając bezpieczeństwo typów za pomocą argumentów.

Przypadki użycia fragmentów.

- Korzystając z komponentu Nawigacja, wiele aplikacji może zarządzać całym swoim układem w ramach jednego działania, a cała nawigacja odbywa się między fragmentami.
- Fragmenty umożliwiają tworzenie wspólnych wzorców układów, takich jak układy wzorzec-szczegóły na tabletach lub wiele kart w ramach tego samego działania.

13. Dowiedz się więcej

- Paprochy
- Odniesienie do klasy fragmentów
- SafeArgs
- Odniesienie do klasy pakietu
- Null Safety w Kotlin
- Widok kontenera fragmentów

Przetestuj komponenty nawigacyjne

1. Zanim zaczniesz

W poprzednich ćwiczeniach z programowania poznałeś nawigację za pomocą komponentu nawigacyjnego. W tym laboratorium programowania dowiesz się, jak testować komponenty nawigacyjne. Należy pamiętać, że różni się to od testowania nawigacji bez użycia komponentów nawigacyjnych.

Warunki wstępne

- Utworzyłeś katalogi testowe w Android Studio.
- Napisałeś testy jednostkowe i oprzyrządowanie w Android Studio.
- Do projektu Androida dodano zależności Gradle.

Czego się nauczysz

- Jak używać testów oprzyrządowania do testowania komponentów nawigacyjnych.
- Jak skonfigurować testy bez powtarzania kodu.

Czego potrzebujesz

- Komputer z zainstalowanym Android Studio.
- Kod rozwiązania dla aplikacji **Words** .

Pobierz kod startowy do tego ćwiczenia z programowania

W tym laboratorium programowania dodasz testy instrumentacji do kodu rozwiązania dla aplikacji **Words** .

URL kodu startowego: https://github.com/google-developer-training/android-basics-kotlin-words-app/tree/main

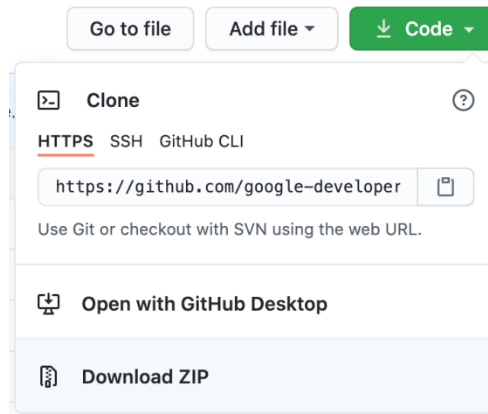
Nazwa modułu z kodem startowym:

main

Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

Pobierz kod

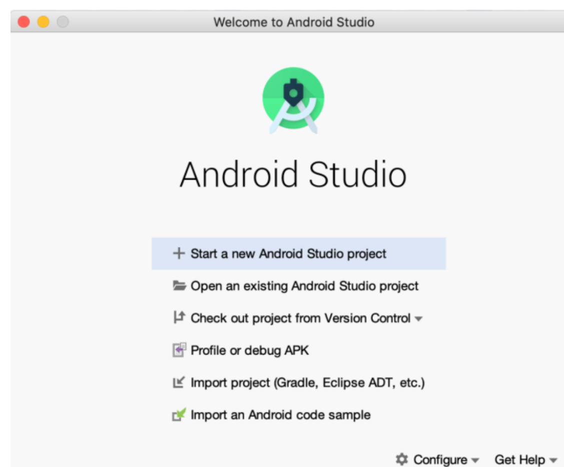
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



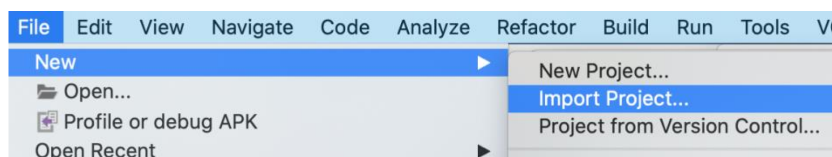
3. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
4. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
5. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

Otwórz projekt w Android Studio

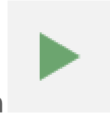
1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz istniejący projekt Android Studio.**



Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Nowy** > **Importuj projekt** .



3. W oknie dialogowym **Importuj projekt** przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.

6. Kliknij przycisk **Uruchom**  , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.
7. Przeglądaj pliki projektu w oknie narzędzia **Projekt** , aby zobaczyć, jak skonfigurowana jest aplikacja.

2. Przegląd aplikacji startowej

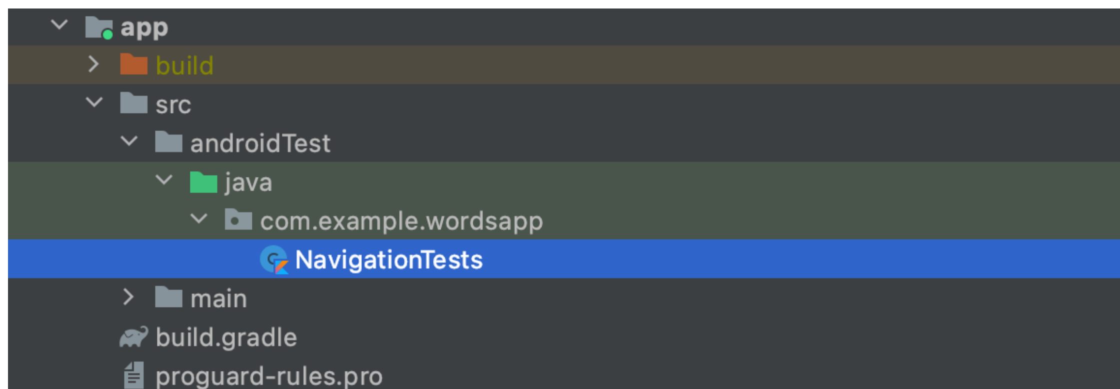
Aplikacja **Words** składa się z ekranu głównego, na którym wyświetlana jest lista, na której każdy element listy jest literą alfabetu. Kliknięcie litery powoduje przejście do ekranu z listą słów zaczynających się od tej litery.

3. Utwórz katalogi testów

W razie potrzeby utwórz katalog testowy oprzyrządowania dla aplikacji Words, tak jak w poprzednich ćwiczeniach z programowania. Jeśli już to zrobiłeś, możesz przejść do Dodaj niezbędne zależności.

4. Utwórz test oprzyrządowania Class

Utwórz nową klasę o nazwie **NavigationTests.kt** w folderze **androidTest** .



5. Dodaj niezbędne zależności

Testowanie składników nawigacji wymaga pewnych określonych zależności Gradle. Dodamy również zależność, która pozwala nam testować fragmenty w bardzo specyficzny sposób. Przejdź do pliku **build.gradle** modułu aplikacji i dodaj następującą zależność:

```
androidTestImplementation 'com.android.support.test.espresso:espresso-contrib:3.0.2'
androidTestImplementation 'androidx.navigation:navigation-testing:2.3.5'

debugImplementation 'androidx.fragment:fragment-testing:1.3.6'
```

Teraz zsynchronizuj swój projekt.

6. Napisz test komponentu nawigacyjnego

1. Określ biegacza testowego, jeśli jeszcze tego nie zrobiłeś.

```
@RunWith(AndroidJUnit4::class)
```

Testowanie komponentów nawigacji różni się od testowania zwykłej nawigacji. Kiedy testujemy zwykłą nawigację, uruchamiamy nawigację na urządzeniu lub emulatorze. Kiedy testujemy komponenty nawigacyjne, w rzeczywistości nie sprawiamy, że urządzenie lub emulator w sposób widoczny nawiguje. Zamiast tego zmuszamy kontroler nawigacyjny do nawigacji bez faktycznej zmiany tego, co jest widoczne na urządzeniu lub emulatorze, a następnie sprawdzamy, czy kontroler nawigacyjny dotarł do właściwego miejsca docelowego.

2. Utwórz funkcję testową o nazwie `navigate_to_words_nav_component()`.
3. Praca z komponentami nawigacyjnymi w testach wymaga pewnej konfiguracji. W `navigate_to_words_nav_component()`metodzie utwórz testową instancję kontrolera nawigacji.

```
val navController = TestNavHostController(
  ApplicationProvider.getApplicationContext()
)
```

4. Komponenty nawigacyjne sterują interfejsem użytkownika za pomocą fragmentów. Istnieje odpowiednik fragmentu `ActivityScenarioRule`, który można wykorzystać do wyizolowania fragmentu do testowania, dlatego wymagana jest zależność specyficzna dla fragmentu. Może to być bardzo przydatne do testowania fragmentu, do którego dotarcie wymaga dużo nawigacji, ponieważ zamiast tego można go uruchomić bez dodatkowego kodu do obsługi nawigowania do niego.

```
val letterListScenario = launchFragmentInContainer<LetterListFragment>(themeResId =
R.style.Theme_Words)
```

Tutaj określamy, że chcemy uruchomić `LetterListFragment`. Musimy przekazać motyw aplikacji, aby komponenty interfejsu użytkownika wiedziały, którego motywu użyć lub test może się zawiesić.

5. Na koniec musimy wyraźnie zadeklarować, którego wykresu nawigacyjnego chcemy, aby kontroler nawigacji używał dla uruchomionego fragmentu.

```
letterListScenario.onFragment { fragment ->

  navController.setGraph(R.navigation.nav_graph)

  Navigation.setViewNavController(fragment.requireView(), navController)
}
```

6. Teraz uruchom zdarzenie, które podpowiada nawigację.

```
onView(withId(R.id.recycler_view))
  .perform(RecyclerViewActions
    .actionOnItemAtPosition<RecyclerView.ViewHolder>(2, click()))
```

Podczas korzystania z tej `launchFragmentInContainer()`metody rzeczywista nawigacja nie jest możliwa, ponieważ kontener nie jest świadomy innych fragmentów lub działań, do których możemy nawigować. Zna tylko fragment, który określiliśmy do uruchomienia w nim. Dlatego po uruchomieniu tego testu na urządzeniu lub emulatorze nie zobaczysz żadnej rzeczywistej nawigacji. Może się to wydawać nieintuicyjne, ale pozwala nam na bardziej bezpośrednie stwierdzenie dotyczące aktualnego celu. Zamiast szukać komponentu UI, o którym wiemy, że wyświetla się na konkretnym ekranie, możemy sprawdzić, czy bieżące miejsce docelowe kontrolera nawigacji ma identyfikator fragmentu, w którym spodziewamy się znaleźć. Takie podejście jest znacznie bardziej niezawodne niż wcześniej.

```
assertEquals(navController.currentDestination?.id, R.id.wordListFragment)
```

Twój test powinien wyglądać mniej więcej tak:

7. Kod rozwiązania

URL kodu rozwiązania: https://github.com/google-developer-training/android-basics-kotlin-words-app

Nazwa modułu z kodem rozwiązania:

nav_test_solution

8. Unikaj powtarzania kodu z adnotacjami

W systemie Android zarówno testy instrumentacji, jak i testy jednostkowe mają funkcję, która pozwala nam ustawić tę samą konfigurację dla każdego testu w klasie bez powtarzania kodu.

Powiedzmy na przykład, że mieliśmy fragment z 10 przyciskami. Każdy przycisk po kliknięciu prowadzi do unikalnego fragmentu.

Gdybyśmy postępowali zgodnie ze wzorcem w powyższym teście, musielibyśmy powtórzyć kod, który wyglądał tak dla każdego z 10 testów (zauważ, że ten kod jest ściśle przykładowy i nie skompiluje się w aplikacji, z którą pracowaliśmy w tym laboratorium):

```
val navController = TestNavHostController(
   ApplicationProvider.getApplicationContext()
)

val exampleFragmentScenario = launchFragmentInContainer<ExampleFragment>(themeResId =
R.style.Theme_Example)

exampleFragmentScenario.onFragment { fragment ->

  navController.setGraph(R.navigation.example_nav_graph)

  Navigation.setViewNavController(fragment.requireView(), navController)
}
```

To dużo kodu do powtórzenia 10 razy. W tym przypadku jednak możemy zaoszczędzić trochę czasu, korzystając z `@Before`adnotacji dostarczonej przez JUnit. Używamy tego, dodając adnotację do metody, w której następnie dostarczamy kod potrzebny do skonfigurowania naszego testu. Metodę możemy nazwać jak nam się podoba, ale powinno to być istotne. Zamiast konfigurować ten sam fragment 10 razy, możemy napisać kod instalacyjny raz w ten sposób:

```
lateinit var navController: TestNavHostController

lateinit var exampleFragmentScenario: FragmentScenario<ExampleFragment>

@Before
fun setup(){
   navController = TestNavHostController(
      ApplicationProvider.getApplicationContext()
   )

   exampleFragmentScenario =  launchFragmentInContainer(themeResId=R.style.Theme_Example)

   exampleFragmentScenario.onFragment { fragment ->

     navController.setGraph(R.navigation.example_nav_graph)

     Navigation.setViewNavController(fragment.requireView(),  navController)
   }
}
```

Ta metoda jest teraz uruchamiana dla każdego testu, który piszemy w tej klasie i możemy uzyskać dostęp do niezbędnych zmiennych z dowolnego testu.

Podobnie, jeśli istnieje kod, który musimy wykonać po każdym teście, możemy użyć `@After`adnotacji. Na przykład `@After`może służyć do oczyszczenia zasobu, którego użyliśmy do naszego testu, lub do testów oprzyrządowania, możemy go użyć do przywrócenia urządzenia do określonego stanu.

JUnit zapewnia również adnotacje `@BeforeClass`i . `@AfterClass`Różnica polega na tym, że metody z tą adnotacją są wykonywane raz, ale wykonany kod nadal stosuje się do każdej metody. Jeśli Twoja konfiguracja lub metody rozłączania zawierają kosztowne operacje, lepiej będzie zamiast tego użyć tych adnotacji. Metody z adnotacjami `@BeforeClass`i `@AfterClass`muszą być umieszczone w obiekcie towarzyszącym i opatrzone adnotacjami `@JvmStatic`. Aby zademonstrować kolejność wykonywania tych adnotacji, spójrzmy na następujący kod:

```kotlin
@RunWith(AndroidJUnit4::class)
class OrderOfTestAnnotations {

    @Before
    fun setupFunction() {
        println("Set up function")
    }

    @Test
    fun test_a() {
        println("Test a")
    }

    @Test
    fun test_b() {
        println("Test b")
    }

    @Test
    fun test_c() {
        println("Test c")
    }

    @After
    fun tearDownFunction() {
        println("Tear down function")
    }

    companion object {
        @BeforeClass
        @JvmStatic
        fun setupClass() {
            println("Set up class")
        }

        @AfterClass
        @JvmStatic
        fun tearDownClass() {
            println("Tear down class")
        }
    }
}
```

Pamiętaj, `@BeforeClass`działa dla klasy, `@Before`działa przed funkcjami, `@After`działa po funkcjach i `@AfterClass`działa dla klasy. Czy możesz przewidzieć, jak będzie to wyglądać?

```
/com.example.wordsapp I/System.out: Set up class
/com.example.wordsapp I/System.out: Set up function
/com.example.wordsapp I/System.out: Test a
/com.example.wordsapp I/System.out: Tear down function
/com.example.wordsapp I/System.out: Set up function
/com.example.wordsapp I/System.out: Test b
/com.example.wordsapp I/System.out: Tear down function
/com.example.wordsapp I/System.out: Set up function
/com.example.wordsapp I/System.out: Test c
/com.example.wordsapp I/System.out: Tear down function
/com.example.wordsapp I/System.out: Tear down class
```

Kolejność wykonywania funkcji to setupClass(), setupFunction(), test_a(), tearDownFunction(), setupFunction(), test_b(), tearDownFunction(),setupFunction(), test_c(), tearDownFunction(), tearDownClass(). Ma to sens, ponieważ @Beforei @Afteruruchom odpowiednio przed i po każdej metodzie. @BeforeClassuruchamia się raz, zanim cokolwiek w klasie zostanie uruchomione i @AfterClassraz uruchomi się po tym, jak wszystko inne w klasie zostanie uruchomione.

9. Gratulacje

W tym ćwiczeniu z programowania możesz:

- Dowiedz się, jak testować komponenty nawigacyjne.
- Dowiedzieliśmy się, jak uniknąć powtarzającego się kodu za pomocą adnotacji @Before, @BeforeClass, @Afteri .@AfterClass

Elementy architektury

Dowiedz się, jak korzystać ze składników systemu Android Jetpack Architecture — zbioru bibliotek, które pomagają w projektowaniu niezawodnych, testowalnych i konserwowalnych aplikacji.

Przechowuj dane w ViewModel

1. Zanim zaczniesz

W poprzednich ćwiczeniach z programowania zapoznałeś się z cyklem życia działań i fragmentów oraz powiązanymi problemami cyklu życia ze zmianami konfiguracji. Jedną z opcji, aby zapisać dane aplikacji, jest zapisanie stanu instancji, ale wiąże się to z własnymi ograniczeniami. W tym laboratorium programowania poznasz niezawodny sposób projektowania aplikacji i zachowywania danych aplikacji podczas zmian konfiguracji, korzystając z bibliotek systemu Android Jetpack.

Biblioteki systemu Android Jetpack to zbiór bibliotek, które ułatwiają tworzenie świetnych aplikacji na Androida. Te biblioteki ułatwiają stosowanie najlepszych praktyk, uwalniają Cię od pisania standardowego kodu i upraszczają złożone zadania, dzięki czemu możesz skupić się na kodzie, na którym Ci zależy, takim jak logika aplikacji.
Komponenty architektury systemu Android są częścią bibliotek systemu Android Jetpack , które ułatwiają projektowanie aplikacji o dobrej architekturze. Składniki architektury zawierają wskazówki dotyczące architektury aplikacji i jest to zalecane najlepsze rozwiązanie.

Architektura aplikacji to zbiór zasad projektowania. Podobnie jak projekt domu, Twoja architektura zapewnia strukturę Twojej aplikacji. Dobra architektura aplikacji może sprawić, że Twój kod będzie solidny, elastyczny, skalowalny i łatwy w utrzymaniu przez wiele lat.

W tym ćwiczeniu z programowania dowiesz się, jak używać `ViewModel`, jednego z komponentów architektury do przechowywania danych aplikacji. Przechowywane dane nie zostaną utracone, jeśli struktura zniszczy i ponownie utworzy działania i fragmenty podczas zmiany konfiguracji lub innych zdarzeń.

Warunki wstępne

- Jak pobrać kod źródłowy z GitHub i otworzyć go w Android Studio.
- Jak stworzyć i uruchomić podstawową aplikację na Androida w Kotlinie, korzystając z działań i fragmentów.
- Wiedza na temat pola tekstowego Materiał i popularnych widżetów interfejsu użytkownika, takich jak `TextView`i `Button`.
- Jak korzystać z powiązania widoku w aplikacji.
- Podstawy działania i cykl życia fragmentów.
- Jak dodać informacje o logowaniu do aplikacji i czytać logi za pomocą **Logcat** w Android Studio.

Czego się nauczysz

- Wprowadzenie do podstaw architektury aplikacji na Androida .
- Jak korzystać z `ViewModel`klasy w swojej aplikacji.

- Jak zachować dane interfejsu użytkownika poprzez zmiany konfiguracji urządzenia przy użyciu ViewModel.
- Właściwości podłoża w Kotlinie.
- Jak korzystać MaterialAlertDialogz biblioteki Material Design Components.

Co zbudujesz

- Gra Rozszyfruj , w której użytkownik może odgadnąć zaszyfrowane słowa.

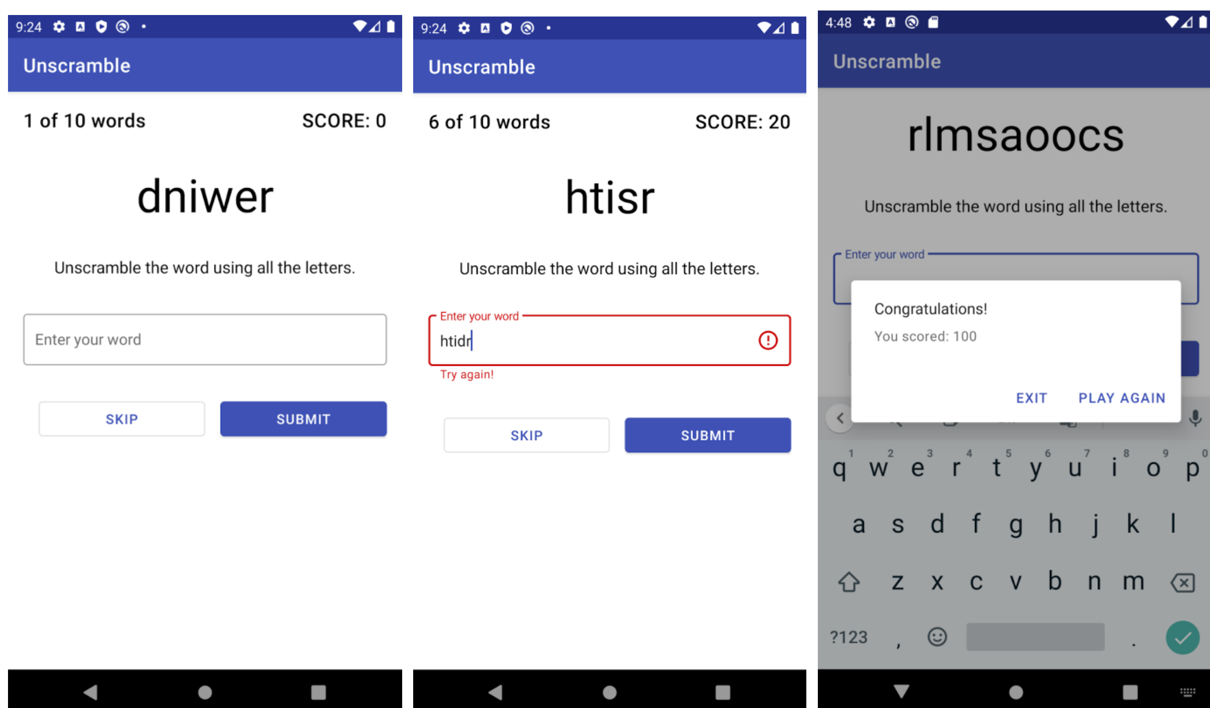
Czego potrzebujesz

- Komputer z zainstalowanym Android Studio.
- Kod startowy aplikacji Unscramble.

2. Przegląd aplikacji startowej

Przegląd gry

Aplikacja Unscramble to jednorazowa gra polegająca na mieszaniu słów. Aplikacja wyświetla jedno zaszyfrowane słowo na raz, a gracz musi je odgadnąć, używając wszystkich liter z zaszyfrowanego słowa. Gracz zdobywa punkty, jeśli słowo jest poprawne, w przeciwnym razie gracz może spróbować dowolną liczbę razy. Aplikacja ma również opcję pominięcia bieżącego słowa. W lewym górnym rogu aplikacja wyświetla liczbę słów, czyli liczbę słów zagranych w bieżącej grze. W grze jest 10 słów.



Pobierz kod startowy

To ćwiczenie z programowania zapewnia kod startowy, który można rozszerzyć o funkcje nauczane w tym ćwiczeniu z programowania. Kod startowy może zawierać kod, który jest Ci zarówno znany, jak i nieznany z poprzednich ćwiczeń z programowania. Więcej o nieznanym kodzie dowiesz się w późniejszych ćwiczeniach z programowania.

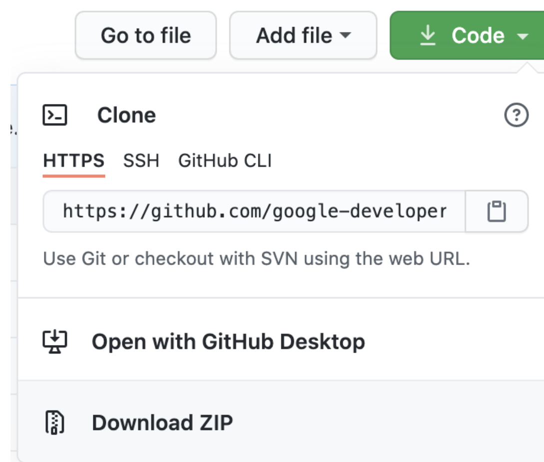
Jeśli używasz kodu startowego z GitHub, pamiętaj, że nazwa folderu to `android-basics-kotlin-unscramble-app-starter`. Wybierz ten folder podczas otwierania projektu w Android Studio.

URL kodu startowego: https://github.com/google-developer-training/android-basics-kotlin-unscramble-app/tree/starter

Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

Pobierz kod

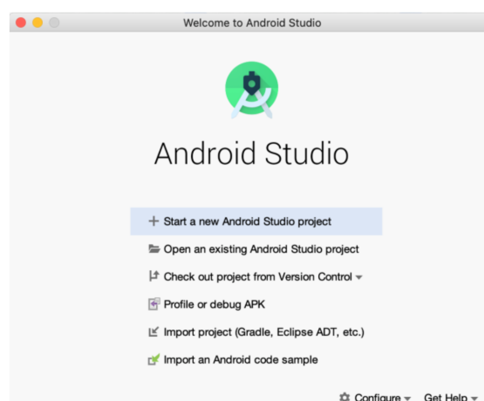
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



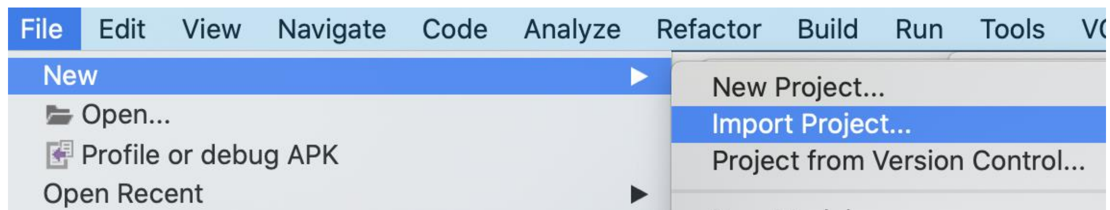
3. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
4. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
5. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

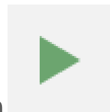
Otwórz projekt w Android Studio

1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz istniejący projekt Android Studio.**



Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Nowy** > **Importuj projekt** .



3. W oknie dialogowym **Importuj projekt** przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.
6. Kliknij przycisk **Uruchom**  , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.
7. Przeglądaj pliki projektu w oknie narzędzia **Projekt** , aby zobaczyć, jak skonfigurowana jest aplikacja.

Przegląd kodu startowego

1. Otwórz projekt za pomocą kodu startowego w Android Studio.
2. Uruchom aplikację na urządzeniu z systemem Android lub w emulatorze.
3. Zagraj w grę, używając kilku słów, dotykając przycisków **Prześlij** i **Pomiń** . Zauważ, że dotknięcie przycisków wyświetla następne słowo i zwiększa liczbę słów.
4. Zauważ, że wynik zwiększa się dopiero po naciśnięciu przycisku **Prześlij** .

Problemy z kodem startowym

Podczas grania w grę mogłeś zaobserwować następujące błędy:

1. Po kliknięciu przycisku **Prześlij** aplikacja nie sprawdza słowa gracza. Gracz zawsze zdobywa punkty.
2. Nie ma możliwości zakończenia gry. Aplikacja pozwala grać więcej niż 10 słów.
3. Ekran gry pokazuje zakodowane słowo, wynik gracza i liczbę słów. Zmień orientację ekranu, obracając urządzenie lub emulator. Zauważ, że bieżące słowo, wynik i liczba słów są stracone, a gra rozpoczyna się od początku.

Główne problemy w aplikacji

Aplikacja startowa nie zapisuje i nie przywraca stanu i danych aplikacji podczas zmian konfiguracji, na przykład w przypadku zmiany orientacji urządzenia.

Możesz rozwiązać ten problem, korzystając z `onSaveInstanceState()`wywołania zwrotnego. Jednak użycie tej `onSaveInstanceState()`metody wymaga napisania dodatkowego kodu, aby zapisać stan w pakiecie i zaimplementować logikę w celu pobrania tego stanu. Ponadto ilość danych, które można przechowywać, jest minimalna.
Możesz rozwiązać te problemy, korzystając ze składników architektury systemu Android , o których poznasz w tej ścieżce.

Przejdź przez kod startowy

Pobrany kod startowy ma wstępnie zaprojektowany układ ekranu gry. W tej ścieżce skupisz się na implementacji logiki gry. Użyjesz komponentów architektury, aby zaimplementować zalecaną architekturę aplikacji i rozwiązać wyżej wymienione problemy. Oto krótki opis niektórych plików na początek.

game_fragment.xml

- Otwórz `res/layout/game_fragment.xml`w widoku **Projekt** .
- Zawiera układ jedynego ekranu w Twojej aplikacji, czyli ekranu gry.
- Ten układ zawiera pole tekstowe na słowo gracza, wraz z `TextViews`wyświetlaniem wyniku i liczby słów. Zawiera również instrukcje i przyciski (**Prześlij** i **Pomiń**), aby zagrać w grę.

główna_aktywność.xml

Definiuje główny układ aktywności za pomocą jednego fragmentu gry.

folder res/values

Znasz pliki zasobów w tym folderze.

- `colors.xml`zawiera kolory motywu używane w aplikacji
- `strings.xml`zawiera wszystkie ciągi znaków, których potrzebuje Twoja aplikacja
- `themes`a `styles`foldery zawierają dostosowanie interfejsu użytkownika wykonane dla Twojej aplikacji

MainActivity.kt

Zawiera domyślny kod wygenerowany przez szablon, aby ustawić widok treści działania jako`main_activity.xml.`

ListaSłów.kt

Ten plik zawiera listę słów używanych w grze, a także stałe określające maksymalną liczbę słów w grze oraz liczbę punktów, które gracz zdobywa za każde poprawne słowo.

UWAGA : Nie zaleca się sztywno kodowania ciągów w kodzie, ciągi powinny być umieszczone w `strings.xml`celu łatwiejszej lokalizacji. Aby zachować prostotę i skupić się na komponentach architektury, w tej aplikacji ciągi są zakodowane na stałe.

GameFragment.kt

To jedyny fragment w Twojej aplikacji, w którym toczy się większość akcji w grze:

- Zmienne są zdefiniowane dla bieżącego zaszyfrowanego słowa (`currentScrambledWord`), liczby słów (`currentWordCount`) i wyniku (`score`).
- Zdefiniowano powiązanie instancji obiektu z dostępem do `game_fragment`wywoływanych widoków .`binding`
- `onCreateView()`Funkcja rozszerza kod `game_fragment`XML układu przy użyciu obiektu powiązania.
- `onViewCreated()`funkcja konfiguruje odbiorniki kliknięcia przycisku i aktualizuje interfejs użytkownika.

- onSubmitWord()jest odbiornikiem kliknięć przycisku **Wyślij** , ta funkcja wyświetla następne zaszyfrowane słowo, czyści pole tekstowe i zwiększa wynik oraz liczbę słów bez sprawdzania poprawności słowa gracza.
- onSkipWord()jest detektorem kliknięć przycisku **Pomiń** , ta funkcja aktualizuje interfejs użytkownika podobnie jak z onSubmitWord()wyjątkiem partytury.
- getNextScrambledWord()to funkcja pomocnicza, która wybiera losowe słowo z listy słów i tasuje zawarte w nim litery.
- restartGame()i exitGame()funkcje służą odpowiednio do ponownego uruchamiania i kończenia gry, użyjesz tych funkcji później.
- setErrorTextField()czyści zawartość pola tekstowego i resetuje stan błędu.
- updateNextWordOnScreen()funkcja wyświetla nowe zaszyfrowane słowo.

3. Dowiedz się więcej o architekturze aplikacji

Architektura dostarcza wskazówek, które pomogą Ci przydzielić obowiązki w Twojej aplikacji między klasami. Dobrze zaprojektowana architektura aplikacji pomaga w skalowaniu aplikacji i rozszerzaniu jej o dodatkowe funkcje w przyszłości. Ułatwia również współpracę zespołową.

Najczęstsze zasady architektoniczne to: oddzielenie problemów i kierowanie interfejsem użytkownika od modelu.

Separacja obaw

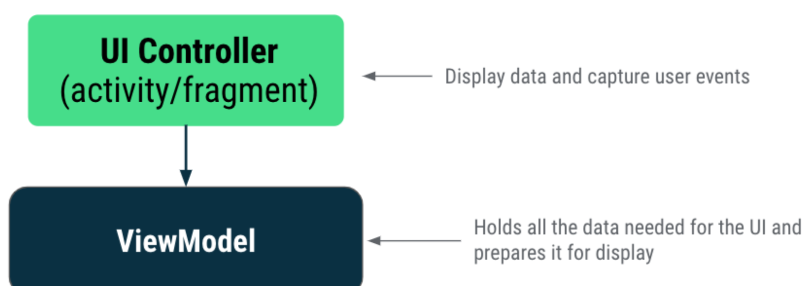
Zasada projektowania separacji obaw mówi, że aplikacja powinna być podzielona na klasy, z których każda ma osobny zakres odpowiedzialności.

Interfejs użytkownika Dysku z modelu

Inną ważną zasadą jest to, że powinieneś kierować swoim interfejsem użytkownika z modelu, najlepiej z modelu trwałego. Modele to składniki odpowiedzialne za obsługę danych aplikacji. Są one niezależne od Viewsskładników i aplikacji w aplikacji, więc cykl życia aplikacji i związane z nią problemy nie mają na nie wpływu.

Główne klasy lub składniki w architekturze systemu Android to Kontroler interfejsu użytkownika (aktywność/fragment) ViewModeli LiveData. RoomTe składniki dbają o pewną złożoność cyklu życia i pomagają uniknąć problemów związanych z cyklem życia. Dowiesz się o LiveDatai Roomw późniejszych ćwiczeniach z programowania.

Ten diagram przedstawia podstawową część architektury:



Kontroler interfejsu użytkownika (aktywność / fragment)

Działania i fragmenty są kontrolerami interfejsu użytkownika. Kontrolery interfejsu użytkownika kontrolują interfejs użytkownika, rysując widoki na ekranie, przechwytując zdarzenia użytkownika i wszystko inne związane z interfejsem użytkownika, z którym użytkownik wchodzi w interakcje. Dane w aplikacji lub wszelkie logiki podejmowania decyzji dotyczące tych danych nie powinny znajdować się w klasach kontrolera interfejsu użytkownika.

System Android może w dowolnym momencie zniszczyć kontrolery interfejsu użytkownika na podstawie określonych interakcji użytkownika lub z powodu warunków systemowych, takich jak mała ilość pamięci. Ponieważ te zdarzenia nie są pod Twoją kontrolą, nie należy przechowywać żadnych danych aplikacji ani stanu w kontrolerach interfejsu użytkownika. Zamiast tego należy dodać logikę podejmowania decyzji dotyczących danych w swoim `ViewModel`.

Na przykład w aplikacji **Unscramble** zaszyfrowane słowo, wynik i liczba słów są wyświetlane we fragmencie (kontroler interfejsu użytkownika). Kod decyzyjny, taki jak obliczanie następnego zaszyfrowanego słowa oraz obliczanie wyniku i liczby słów, powinien znajdować się w Twoim `ViewModel`.

ZobaczModel

Jest ViewModelto model danych aplikacji wyświetlany w widokach. Modele to składniki odpowiedzialne za obsługę danych aplikacji. Pozwalają Twojej aplikacji przestrzegać zasady architektury, sterując interfejsem użytkownika z modelu.
Przechowuje dane związane z ViewModelaplikacją, które nie są niszczone, gdy działanie lub fragment zostanie zniszczony i odtworzony przez platformę Android. `ViewModel`obiekty są automatycznie zachowywane (nie są niszczone, jak aktywność lub instancja fragmentu) podczas zmian konfiguracji, dzięki czemu przechowywane w nich dane są natychmiast dostępne dla następnego działania lub instancji fragmentu.

Aby zaimplementować `ViewModel`w swojej aplikacji, rozszerz `ViewModel`klasę, która pochodzi z biblioteki komponentów architektury, i przechowuj dane aplikacji w tej klasie.

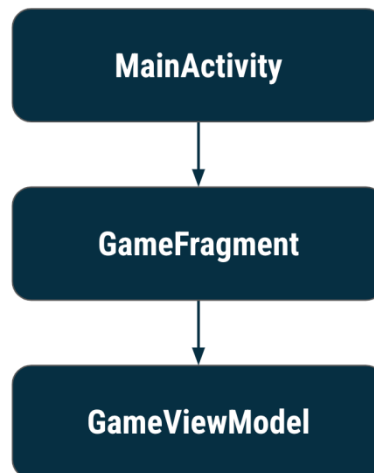
Podsumowując:

Obowiązki fragmentu / czynności (kontrolera UI)	ViewModelobowiązki
Działania i fragmenty odpowiadają za rysowanie widoków i danych na ekranie oraz reagowanie na zdarzenia użytkownika.	ViewModelodpowiada za przechowywanie i przetwarzanie wszystkich danych potrzebnych do IU. Nigdy nie powinien uzyskiwać dostępu do hierarchii widoków (takich jak obiekt powiązania widoku) ani zawierać odniesienia do działania lub fragmentu.

4. Dodaj ViewModel

W tym zadaniu dodajesz a `ViewModel`do swojej aplikacji, aby przechowywać dane aplikacji (zaszyfrowane słowa, liczba słów i wynik).

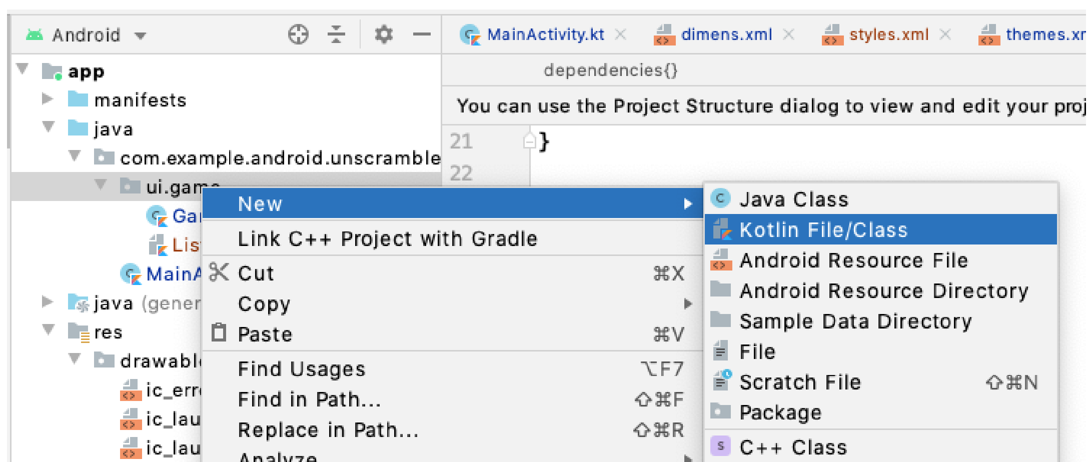
Twoja aplikacja zostanie zaprojektowana w następujący sposób. MainActivityzawiera GameFragment, a GameFragmentuzyska dostęp do informacji o grze z GameViewModel.



1. W oknie **Androida** Android Studio w folderze **Gradle Scripts** otwórz plik build.gradle(Module:Unscramble.app).
2. Aby użyć w swojej aplikacji, sprawdź, czy w bloku ViewModelznajduje się zależność biblioteki ViewModel . dependenciesTen krok jest już zrobiony za Ciebie. W zależności od najnowszej wersji biblioteki numer wersji biblioteki w wygenerowanym kodzie może być inny.

   ```
   // ViewModel
   implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1'
   ```

   Zaleca się, aby zawsze używać najnowszej wersji biblioteki, niezależnie od wersji wymienionej w ćwiczeniu z programowania.
3. Utwórz nowy plik klasy Kotlin o nazwie GameViewModel. W oknie **Androida** kliknij prawym przyciskiem myszy folder **ui.game** . Wybierz **Nowy** > **Plik/Klasa Kotlina** .



4. Nadaj mu nazwę GameViewModeli wybierz z listy **Klasa .**
5. Zmiana GameViewModeldo podklasy z ViewModel. ViewModeljest klasą abstrakcyjną, więc musisz ją rozszerzyć, aby używać jej w swojej aplikacji. Zobacz GameViewModeldefinicję klasy poniżej.

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

Dołącz ViewModel do fragmentu

Aby skojarzyć a ViewModelz kontrolerem UI (aktywność / fragment), utwórz odwołanie (obiekt) do ViewModelwnętrza kontrolera UI.

W tym kroku utworzysz instancję obiektu GameViewModelwewnątrz odpowiedniego kontrolera interfejsu użytkownika, czyli GameFragment.

1. U góry GameFragmentklasy dodaj właściwość type GameViewModel.
2. Zainicjuj GameViewModelużycie by viewModels()delegata właściwości Kotlin. Więcej na ten temat dowiesz się w następnym rozdziale.

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

3. Jeśli pojawi się monit Android Studio, zaimportuj androidx.fragment.app.viewModels.

Pełnomocnik ds. nieruchomości Kotlina

W Kotlinie każda varwłaściwość mutable ( ) ma domyślne funkcje pobierające i ustawiające automatycznie generowane dla niej. Funkcje ustawiające i pobierające są wywoływane podczas przypisywania wartości lub odczytywania wartości właściwości.

W przypadku właściwości tylko do odczytu (val) różni się nieco od właściwości mutowalnej. Tylko funkcja pobierająca jest generowana domyślnie. Ta funkcja pobierająca jest wywoływana podczas odczytywania wartości właściwości tylko do odczytu.

Delegacja własności w Kotlinie pomaga przekazać odpowiedzialność pobierającego setera innej klasie.

Ta klasa (zwana *klasą delegata*) udostępnia funkcje pobierające i ustawiające właściwości oraz obsługuje jej zmiany.

Właściwość delegata jest definiowana za pomocą byklauzuli i instancji klasy delegata:

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

W Twojej aplikacji, jeśli zainicjujesz model widoku przy użyciu domyślnego GameViewModelkonstruktora, jak poniżej:

```
private val viewModel = GameViewModel()
```

Następnie aplikacja utraci stan viewModelodniesienia, gdy urządzenie przejdzie przez zmianę konfiguracji. Na przykład, jeśli obrócisz urządzenie, działanie zostanie zniszczone i utworzone ponownie, a otrzymasz nową instancję modelu widoku ze stanem początkowym.

Zamiast tego użyj podejścia delegata właściwości i deleguj odpowiedzialność za viewModelobiekt do oddzielnej klasy o nazwie viewModels. Oznacza to, że kiedy uzyskujesz dostęp

do viewModelobiektu, jest on obsługiwany wewnętrznie przez klasę delegata, viewModels. Klasa delegata tworzy viewModelobiekt dla Ciebie przy pierwszym dostępie i zachowuje jego wartość poprzez zmiany konfiguracji i zwraca wartość na żądanie.

5. Przenieś dane do ViewModel

Oddzielenie danych interfejsu użytkownika aplikacji od kontrolera interfejsu użytkownika (twoje Activity/ Fragmentklasy) pozwala lepiej przestrzegać zasady pojedynczej odpowiedzialności, którą omówiliśmy powyżej. Twoje działania i fragmenty są odpowiedzialne za rysowanie widoków i danych na ekranie, podczas gdy Ty ViewModelodpowiadasz za przechowywanie i przetwarzanie wszystkich danych potrzebnych do interfejsu użytkownika.

W tym zadaniu przenosisz zmienne danych z klasy GameFragmentdo GameViewModelklasy.

1. Przenieś zmienne danych score, currentWordCount, currentScrambledWorddo GameViewModelklasy.

```
class GameViewModel : ViewModel() {

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

2. Zwróć uwagę na błędy dotyczące nierozwiązanych odwołań. Dzieje się tak, ponieważ właściwości są prywatne dla ViewModelkontrolera interfejsu użytkownika i nie są dostępne dla nich. Następnie naprawisz te błędy.

   Aby rozwiązać ten problem, nie można wprowadzić modyfikatorów widoczności właściwości public— dane nie powinny być edytowalne przez inne klasy. Jest to ryzykowne, ponieważ klasa zewnętrzna może zmienić dane w nieoczekiwany sposób, niezgodny z regułami gry określonymi w modelu widoku. Na przykład klasa zewnętrzna może zmienić wartość scorena ujemną.

   Wewnątrz ViewModel, dane powinny być edytowalne, więc powinny być privatei var. Z zewnątrz ViewModeldane powinny być czytelne, ale nie edytowalne, więc dane powinny być widoczne jako publici val. Aby osiągnąć to zachowanie, Kotlin ma funkcję zwaną backing property .

Właściwość podkładowa

Właściwość backing umożliwia zwrócenie czegoś z metody pobierającej innego niż dokładny obiekt.

Nauczyłeś się już, że dla każdej właściwości framework Kotlin generuje gettery i settery.

W przypadku metod pobierających i ustawiających można przesłonić jedną lub obie te metody i zapewnić własne zachowanie niestandardowe. Aby zaimplementować właściwość kopii zapasowej, zastąpisz metodę pobierającą, aby zwrócić wersję danych tylko do odczytu. Przykład właściwości podkładu:

```
// 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
```

Rozważmy przykład, w Twojej aplikacji chcesz, aby dane aplikacji były prywatne dla ViewModel:

W ViewModelklasie:

- Właściwość _countjest privatezmienna. W związku z tym jest dostępny i edytowalny tylko w ramach ViewModelklasy. Konwencja jest taka, aby poprzedzić privatewłaściwość podkreśleniem.

Poza ViewModelzajęciami:

- Domyślny modyfikator widoczności w Kotlin to public, więc countjest publiczny i dostępny z innych klas, takich jak kontrolery interfejsu użytkownika. Ponieważ get()zastępowana jest tylko metoda, ta właściwość jest niezmienna i tylko do odczytu. Gdy klasa zewnętrzna uzyskuje dostęp do tej właściwości, zwraca wartość _counti jej wartości nie można modyfikować. Chroni to dane aplikacji wewnątrz ViewModelprzed niechcianymi i niebezpiecznymi zmianami przez klasy zewnętrzne, ale pozwala zewnętrznym rozmówcom na bezpieczny dostęp do ich wartości.

Dodaj właściwość kopii zapasowej do currentScrambledWord

1. W GameViewModelzmień currentScrambledWorddeklarację, aby dodać właściwość podkładu. Teraz _currentScrambledWordjest dostępny i edytowalny tylko w ramach GameViewModel. Kontroler interfejsu użytkownika GameFragmentmoże odczytać swoją wartość za pomocą właściwości tylko do odczytu, currentScrambledWord.

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

2. W GameFragmentprogramie zaktualizuj metodę tak, updateNextWordOnScreen()aby używała właściwości tylko do odczytu viewModel, currentScrambledWord.

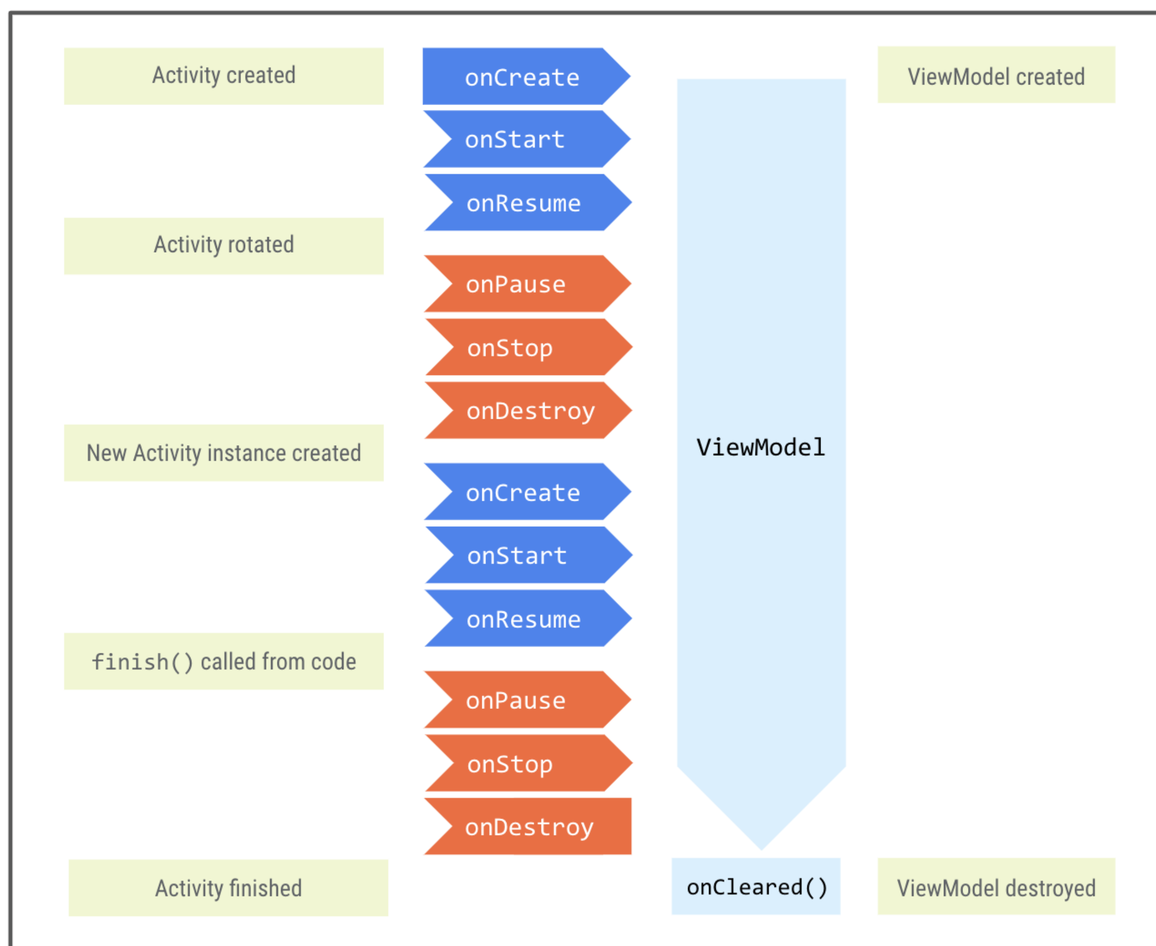
```
private fun updateNextWordOnScreen() {
   binding.textViewUnscrambledWord.text = viewModel.currentScrambledWord
}
```

3. W GameFragmentprogramie usuń kod wewnątrz metod onSubmitWord()i onSkipWord(). Te metody zaimplementujesz później. Teraz powinieneś być w stanie skompilować kod bez błędów.

Ostrzeżenie : nigdy nie ujawniaj zmiennych pól danych ze swojej ViewModel— upewnij się, że te dane nie mogą być modyfikowane z innej klasy. Zmienne dane wewnątrz ViewModelpowinny być zawsze private.

6. Cykl życia ViewModelu

Struktura utrzymuje przy ViewModelżyciu tak długo, jak żywy jest zakres działania lub fragmentu. A ViewModelnie zostanie zniszczony, jeśli jego właściciel zostanie zniszczony w wyniku zmiany konfiguracji, takiej jak obracanie ekranu. Nowa instancja właściciela ponownie łączy się z istniejącą ViewModelinstancją, co ilustruje poniższy diagram:



Zrozumienie cyklu życia ViewModel

Dodaj logowanie w GameViewModeli GameFragment, aby lepiej zrozumieć cykl życia ViewModel.

1. GameViewModel.ktDodaj blok z initinstrukcją log.

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


  ...
}
```

Kotlin udostępnia blok inicjujący (znany również jako `init`blok) jako miejsce na kod konfiguracji początkowej potrzebny podczas inicjalizacji instancji obiektu. Bloki inicjatora są poprzedzone `init`słowem kluczowym, po którym następują nawiasy klamrowe `{}`. Ten blok kodu jest uruchamiany podczas pierwszego tworzenia i inicjowania wystąpienia obiektu.

2. W `GameViewModel`klasie nadpisz [`onCleared()`]metodę. Jest `ViewModel`niszczony po odłączeniu skojarzonego fragmentu lub po zakończeniu działania. Tuż przed `ViewModel`zniszczeniem następuje `onCleared()`wywołanie zwrotne.
3. Dodaj oświadczenie dziennika w środku, `onCleared()`aby śledzić `GameViewModel`cykl życia.

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

4. W `GameFragment`inside `onCreateView()`, po otrzymaniu odwołania do obiektu wiążącego, dodaj instrukcję log, aby zarejestrować utworzenie fragmentu. Callback `onCreateView()`zostanie wyzwolony, gdy fragment zostanie utworzony po raz pierwszy, a także za każdym razem, gdy zostanie ponownie utworzony dla wszelkich zdarzeń, takich jak zmiany konfiguracji.

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

5. W `GameFragment`programie zastąp `onDetach()`metodę wywołania zwrotnego, która zostanie wywołana, gdy odpowiednie działanie i fragment zostaną zniszczone.

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

6. W Android Studio uruchom aplikację, otwórz okno **Logcat** i przefiltruj na `GameFragment`. Zauważ, że `GameFragment`i `GameViewModel`są tworzone.

```
com.example.android.unscramble D/GameFragment: GameFragment created/re-created!
com.example.android.unscramble D/GameFragment: GameViewModel created!
```

7. Włącz ustawienie automatycznego obracania na swoim urządzeniu lub emulatorze i kilkakrotnie zmień orientację ekranu. Jest `GameFragment`niszczony i odtwarzany za każdym razem, ale `GameViewModel`jest tworzony tylko raz i nie jest odtwarzany ani niszczony przy każdym wywołaniu.

```
com.example.android.unscramble D/GameFragment: GameFragment created/re-created!
com.example.android.unscramble D/GameFragment: GameViewModel created!
com.example.android.unscramble D/GameFragment: GameFragment destroyed!
com.example.android.unscramble D/GameFragment: GameFragment created/re-created!
```

```
com.example.android.unscramble D/GameFragment: GameFragment destroyed!
com.example.android.unscramble D/GameFragment: GameFragment created/re-created!
com.example.android.unscramble D/GameFragment: GameFragment destroyed!
com.example.android.unscramble D/GameFragment: GameFragment created/re-created!
com.example.android.unscramble D/GameFragment: GameFragment destroyed!
com.example.android.unscramble D/GameFragment: GameFragment created/re-created!
```

8. Wyjdź z gry lub wyjdź z aplikacji za pomocą strzałki wstecz. Zniszczone GameViewModeli wywołanie zwrotne onCleared(). Jest GameFragmentzniszczony.

```
com.example.android.unscramble D/GameFragment: GameViewModel destroyed!
com.example.android.unscramble D/GameFragment: GameFragment destroyed!
```

7. Wypełnij ViewModel

W tym zadaniu dodatkowo wypełniasz GameViewModelmetody pomocników w celu uzyskania następnego słowa, walidacji słowa gracza w celu zwiększenia wyniku i sprawdzenia liczby słów, aby zakończyć grę.

Późna inicjalizacja

Zazwyczaj, gdy deklarujesz zmienną, podajesz jej wartość początkową z góry. Jeśli jednak nie jesteś jeszcze gotowy do przypisania wartości, możesz zainicjować ją później. Aby późno zainicjować właściwość w Kotlinie, używasz słowa kluczowego lateinit, co oznacza późną inicjalizację. Jeśli gwarantujesz, że zainicjujesz właściwość przed jej użyciem, możesz zadeklarować właściwość za pomocą lateinit. Pamięć nie jest przydzielana do zmiennej, dopóki nie zostanie zainicjowana. Jeśli spróbujesz uzyskać dostęp do zmiennej przed jej zainicjowaniem, aplikacja ulegnie awarii.

Zdobądź następne słowo

Utwórz getNextWord()metodę w GameViewModelklasie z następującą funkcjonalnością:

- Pobierz losowe słowo z allWordsListi przypisz je docurrentWord.
- Utwórz zaszyfrowane słowo, mieszając litery w currentWordi przypisz je docurrentScrambledWord
- Zajmij się przypadkiem, w którym zaszyfrowane słowo jest takie samo jak niezaszyfrowane słowo.
- Upewnij się, że podczas gry nie pokazujesz dwa razy tego samego słowa.

Zaimplementuj w klasie następujące kroki GameViewModel:

1. GameViewModel,Dodaj nową zmienną klasową typu MutableList<String>o nazwie wordsList, aby przechowywać listę słów, których używasz w grze, aby uniknąć powtórzeń .
2. Dodaj kolejną zmienną klasy wywoływaną currentWorddo przechowywania słowa, które gracz próbuje rozszyfrować. Użyj lateinitsłowa kluczowego, ponieważ zainicjujesz tę właściwość później.

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

3. Dodaj nową privatemetodę o nazwie getNextWord(), powyżej initbloku, bez parametrów, która nic nie zwraca.
4. Pobierz losowe słowo z allWordsListi przypisz je do currentWord.

```
private fun getNextWord() {
  currentWord = allWordsList.random()
}
```

5. W getNextWord()programie , przekonwertuj currentWordciąg na tablicę znaków i przypisz go do nowego valwywołanego tempWord. Aby zaszyfrować słowo, przetasuj znaki w tej tablicy za pomocą metody Kotlin, shuffle().

```
val tempWord = currentWord.toCharArray()
tempWord.shuffle()
```

An Arrayjest podobny do List, ale po zainicjowaniu ma stały rozmiar. An Arraynie może zwiększać ani zmniejszać swojego rozmiaru (trzeba skopiować tablicę, aby zmienić jej rozmiar), podczas gdy a Listma add()i remove()działa, dzięki czemu może zwiększać lub zmniejszać rozmiar.

6. Czasami przetasowana kolejność znaków jest taka sama jak oryginalnego słowa. Dodaj następującą whilepętlę wokół wywołania tasowania, aby kontynuować pętlę, aż zaszyfrowane słowo nie będzie takie samo jak oryginalne słowo.

```
while (String(tempWord).equals(currentWord, false)) {
   tempWord.shuffle()
}
```

7. Dodaj if-elseblok, aby sprawdzić, czy słowo zostało już użyte. Jeśli wordsListzawiera currentWord, zadzwoń getNextWord(). Jeśli nie, zaktualizuj wartość _currentScrambledWordnowo zaszyfrowanym słowem, zwiększ liczbę słów i dodaj nowe słowo do wordsList.

```
if (wordsList.contains(currentWord)) {
   getNextWord()
} else {
   _currentScrambledWord = String(tempWord)
   ++_currentWordCount
   wordsList.add(currentWord)
}
```

8. Oto ukończona getNextWord()metoda w celach informacyjnych.

```
/*
* 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)
    }
}
```

Późna inicjalizacja prąduScrambledWord

Teraz stworzyłeś `getNextWord()`metodę, aby uzyskać następne zaszyfrowane słowo. Zadzwonisz do niego, gdy `GameViewModel`zostanie zainicjowany po raz pierwszy. Użyj `init`bloku, aby zainicjować `lateinit`właściwości w klasie, takie jak bieżące słowo. W rezultacie pierwsze słowo wyświetlone na ekranie będzie zakodowanym słowem zamiast **test** .

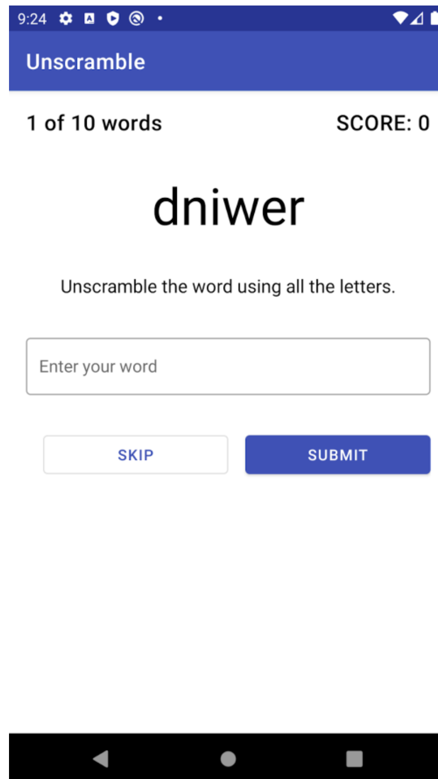
1. Uruchom aplikację. Zauważ, że pierwsze słowo to zawsze „ **test** ”.
2. Aby wyświetlić zaszyfrowane słowo na początku aplikacji, musisz wywołać `getNextWord()`metodę, która z kolei aktualizuje `currentScrambledWord`. Wywołaj metodę `getNextWord()`wewnątrz `init`bloku `GameViewModel`.

```
init {
    Log.d("GameFragment", "GameViewModel created!")
    getNextWord()
}
```

3. Dodaj `lateinit`modyfikator do `_currentScrambledWord`właściwości. Dodaj jawną wzmiankę o typie danych `String`, ponieważ nie podano wartości początkowej.

```
private lateinit var _currentScrambledWord: String
```

4. Uruchom aplikację. Zwróć uwagę, że podczas uruchamiania aplikacji wyświetlane jest nowe zaszyfrowane słowo. Wspaniały!



Dodaj metodę pomocniczą

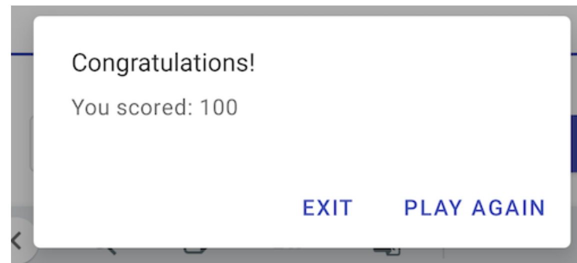
Następnie dodaj metodę pomocniczą, aby przetwarzać i modyfikować dane wewnątrz ViewModel. Użyjesz tej metody w późniejszych zadaniach.

1. W GameViewModelklasie dodaj kolejną metodę o nazwie nextWord().Pobierz następne słowo z listy i zwróć true, jeśli liczba słów jest mniejsza niż *MAX_NO_OF_WORDS*.

```
/*
* 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
}
```

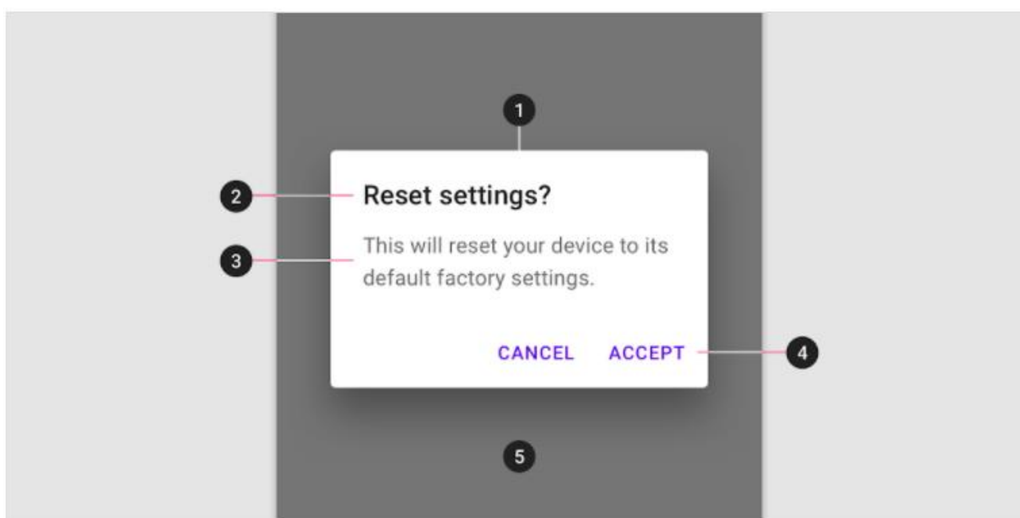
8. Dialogi

W kodzie startowym gra nigdy się nie kończyła, nawet po rozegraniu 10 słów. Zmodyfikuj swoją aplikację tak, aby po przejściu przez użytkownika 10 słów gra się skończyła i wyświetliło się okno dialogowe z końcowym wynikiem. Dasz również użytkownikowi możliwość ponownego zagrania lub wyjścia z gry.



Po raz pierwszy dodasz okno dialogowe do aplikacji. Okno dialogowe to małe okno (ekran), które zachęca użytkownika do podjęcia decyzji lub wprowadzenia dodatkowych informacji. Zwykle okno dialogowe nie wypełnia całego ekranu i wymaga od użytkowników podjęcia działania, zanim będą mogli kontynuować. Android udostępnia różne typy okien dialogowych. W tym ćwiczeniu z programowania dowiesz się o oknach dialogowych alertów.

Anatomia okna dialogowego alertu



1. Okno dialogowe alertu
2. Tytuł (opcjonalnie)
3. Wiadomość
4. Przyciski tekstowe

Zaimplementuj okno dialogowe wyniku końcowego

Użyj [MaterialAlertDialog]biblioteki Material Design Components, aby dodać do aplikacji okno dialogowe zgodne z wytycznymi dotyczącymi materiałów. Ponieważ okno dialogowe jest powiązane z interfejsem użytkownika, `GameFragment`będzie odpowiedzialne za tworzenie i wyświetlanie okna dialogowego końcowego wyniku.

1. Najpierw dodaj do `score`zmiennej właściwość zapasową. W `GameViewModel`programie zmień `score`deklarację zmiennej na następującą.

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

2. W `GameFragment`programie dodaj prywatną funkcję o nazwie `showFinalScoreDialog()`. Aby utworzyć `MaterialAlertDialog`, użyj `MaterialAlertDialogBuilder`klasy do zbudowania części okna dialogowego krok po kroku. Wywołaj [MaterialAlertDialogBuilder]konstruktor przekazujący treść za

pomocą metody fragmentu requireContext(). Metoda requireContext()zwraca wartość inną niż null Context.

```
/*
* Creates and shows an AlertDialog with the final score.
*/
private fun showFinalScoreDialog() {
  MaterialAlertDialogBuilder(requireContext())
}
```

Jak sama nazwa wskazuje, Contextodnosi się do kontekstu lub aktualnego stanu aplikacji, działania lub fragmentu. Zawiera informacje dotyczące czynności, fragmentu lub aplikacji. Zwykle służy do uzyskiwania dostępu do zasobów, baz danych i innych usług systemowych. W tym kroku przekazujesz kontekst fragmentu, aby utworzyć okno dialogowe alertu.

Jeśli pojawi się monit Android Studio, import com.google.android.material.dialog.MaterialAlertDialogBuilder.

3. Dodaj kod, aby ustawić tytuł w oknie dialogowym alertu, użyj zasobu ciągu z strings.xml.

```
MaterialAlertDialogBuilder(requireContext())
  .setTitle(getString(R.string.congratulations))
```

4. Ustaw komunikat tak, aby pokazywał końcowy wynik, użyj wersji tylko do odczytu zmiennej score (viewModel.score), dodanej wcześniej.

```
  .setMessage(getString(R.string.you_scored, viewModel.score))
```

5. Nie możesz anulować okna dialogowego alertu po naciśnięciu klawisza Wstecz, używając setCancelable()metody i przekazując false.

```
  .setCancelable(false)
```

6. Dodaj dwa przyciski tekstowe **EXIT** i **PLAY AGAIN** , korzystając z metod setNegativeButton()i setPositiveButton(). Zadzwoń exitGame()i restartGame()odpowiednio z lambd.

```
  .setNegativeButton(getString(R.string.exit)) { _, _ ->
     exitGame()
  }
  .setPositiveButton(getString(R.string.play_again)) { _, _ ->
     restartGame()
  }
```

Ta składnia może być dla Ciebie nowa, ale jest to skrót od tego, setNegativeButton(getString(R.string.exit), { _, _ -> exitGame() })gdzie setNegativeButton()metoda przyjmuje dwa parametry: a Stringi funkcję, DialogInterface.OnClickListener()którą można wyrazić jako lambda. Kiedy ostatni przekazywany argument jest funkcją, możesz umieścić wyrażenie lambda *poza* nawiasami. Nazywa się to *składnią końcowej lambda* . Oba sposoby pisania kodu (z lambdą wewnątrz lub na zewnątrz nawiasów) są dopuszczalne. To samo dotyczy setPositiveButtonfunkcji.

7. Na koniec dodaj show(), co spowoduje utworzenie, a następnie wyświetlenie okna dialogowego alertu.

```
   .show()
```

8. Oto pełna showFinalScoreDialog()metoda odniesienia.

```
/*
* 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()
}
```

9. Zaimplementuj OnClickListener dla przycisku Prześlij

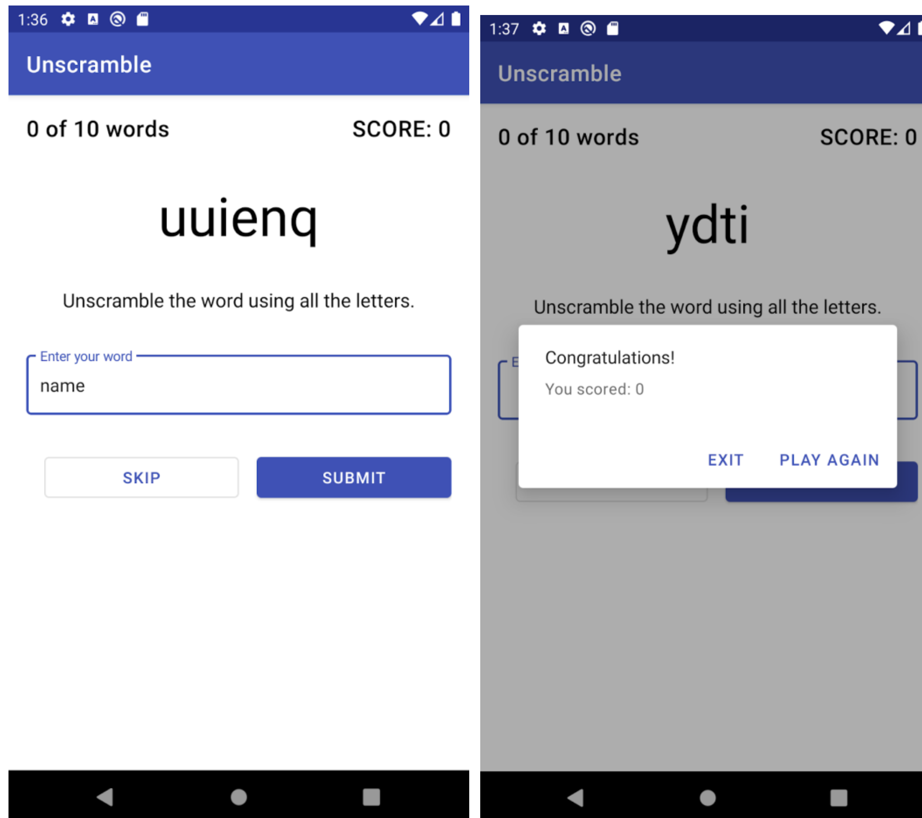
W tym zadaniu ViewModelużyjesz dodanego okna dialogowego i alertu, aby zaimplementować logikę gry dla odbiornika kliknięcia przycisku **Prześlij .**

Wyświetl zaszyfrowane słowa

1. Jeśli jeszcze tego nie zrobiłeś, w GameFragmentprogramie usuń kod znajdujący się wewnątrz onSubmitWord(), który jest wywoływany po dotknięciu przycisku **Prześlij .**
2. Dodaj kontrolę zwracanej wartości viewModel.nextWord()metody. Jeśli truedostępne jest inne słowo, zaktualizuj zaszyfrowane słowo na ekranie za pomocą updateNextWordOnScreen(). W przeciwnym razie gra jest skończona, więc wyświetl okno dialogowe ostrzeżenia z końcowym wynikiem.

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

3. Uruchom aplikację! Zagraj w kilka słów. Pamiętaj, że nie zaimplementowałeś jeszcze przycisku **Pomiń** , więc nie możesz pominąć słowa.
4. Zauważ, że pole tekstowe nie jest aktualizowane, więc gracz musi ręcznie usunąć poprzednie słowo. Ostateczny wynik w oknie alertu to zawsze zero. Naprawisz te błędy w kolejnych krokach.



Dodaj metodę pomocnika, aby sprawdzić poprawność słowa gracza

1. W GameViewModelprogramie dodaj nową metodę prywatną wywołaną increaseScore()bez parametrów i bez zwracanej wartości. Zwiększ scorezmienną o *SCORE_INCREASE*.

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

2. W programie GameViewModeldodaj wywołaną metodę pomocniczą, isUserWordCorrect()która zwraca a Booleani przyjmuje Stringjako parametr słowo gracza.
3. W isUserWordCorrect()popraw słowo gracza i zwiększ wynik, jeśli zgadywanie jest poprawne. Spowoduje to zaktualizowanie końcowego wyniku w oknie alertu.

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

Zaktualizuj pole tekstowe

Pokaż błędy w polu tekstowym

W przypadku pól tekstowych Materiał [TextInputLayout]ma wbudowaną funkcję wyświetlania komunikatów o błędach. Na przykład w poniższym polu tekstowym zmienia się kolor etykiety, wyświetlana jest ikona błędu, wyświetlany jest komunikat o błędzie itd.

Aby wyświetlić błąd w polu tekstowym, możesz ustawić komunikat o błędzie dynamicznie w kodzie lub statycznie w pliku układu. Przykład ustawiania i resetowania błędu w kodzie pokazano poniżej:

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

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

W kodzie startowym znajdziesz `setErrorTextField(error: Boolean)`już zdefiniowaną metodę pomocnika, która pomoże Ci ustawić i zresetować błąd w polu tekstowym. Wywołaj tę metodę z `true`lub `false`jako parametr wejściowy w zależności od tego, czy chcesz, aby błąd był wyświetlany w polu tekstowym, czy nie.

Fragment kodu w kodzie startowym

```
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
  }
}
```

W tym zadaniu implementujesz metodę `onSubmitWord()`. Po przesłaniu słowa potwierdź przypuszczenie użytkownika, porównując z oryginalnym słowem. Jeśli słowo jest poprawne, przejdź do następnego słowa (lub pokaż okno dialogowe, jeśli gra się skończyła). Jeśli słowo jest nieprawidłowe, pokaż błąd w polu tekstowym i pozostań przy bieżącym słowie.

1. Na `GameFragment`,początku `onSubmitWord()`stwórz `val`tzw `playerWord`. Zapisz w nim słowo gracza, wyodrębniając je z pola tekstowego w `binding`zmiennej.

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

```
}
```

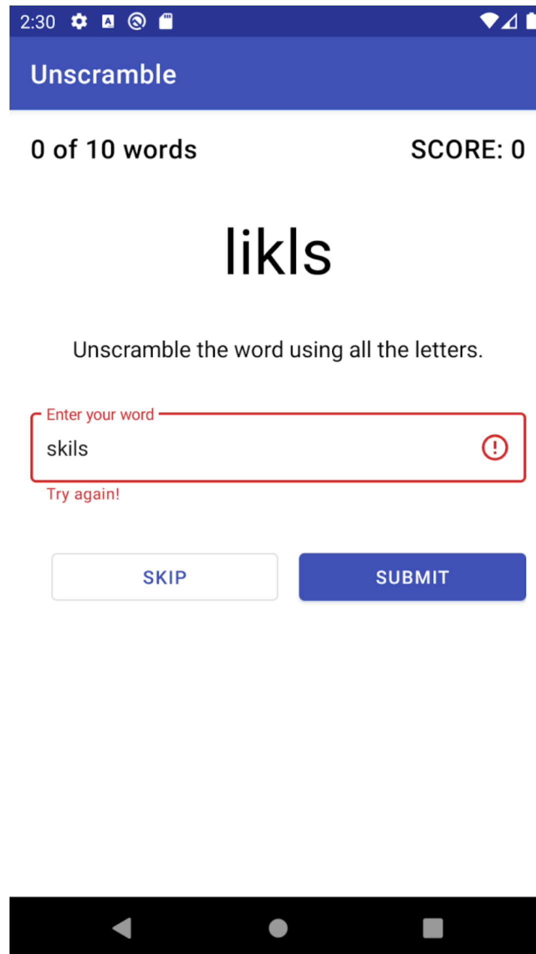
2. W onSubmitWord(), pod deklaracją playerWord, potwierdź słowo gracza. Dodaj ifoświadczenie, aby sprawdzić słowo gracza za pomocą isUserWordCorrect()metody, przekazując playerWord.
3. Wewnątrz ifbloku zresetuj pole tekstowe, wywołaj setErrorTextFieldprzekazywanie false.
4. Przenieś istniejący kod wewnątrz ifbloku.

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

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

5. Jeśli słowo użytkownika jest nieprawidłowe, pokaż komunikat o błędzie w polu tekstowym. Dodaj elseblok do powyższego ifbloku i wywołaj setErrorTextField()przekazywanie true. Twoja ukończona onSubmitWord()metoda powinna wyglądać tak:

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

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

6. Uruchom swoją aplikację. Zagraj w kilka słów. jeśli słowo gracza jest poprawne, słowo jest usuwane po kliknięciu przycisku **Prześlij** , w przeciwnym razie komunikat „Spróbuj ponownie!” jest wyświetlany. Zauważ, że przycisk **Pomiń** nadal nie działa. Dodasz tę implementację w następnym zadaniu.



10. Zaimplementuj przycisk Pomiń

W tym zadaniu dodajesz implementację, onSkipWord()która obsługuje po kliknięciu przycisku **Pomiń .**

1. Podobnie jak onSubmitWord(), dodaj warunek w onSkipWord()metodzie. Jeśli true, wyświetl słowo na ekranie i zresetuj pole tekstowe. Jeśli falsew tej rundzie nie ma już słów, pokaż okno dialogowe ostrzeżenia z końcowym wynikiem.

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

```
}
```

2. Uruchom swoją aplikację. Zagrać w grę. Zwróć uwagę, że przyciski **Pomiń** i **Prześlij** działają zgodnie z przeznaczeniem. Doskonały!

11. Sprawdź, czy ViewModel zachowuje dane

W tym zadaniu dodaj logowanie, `GameFragment`aby zaobserwować, że dane aplikacji są zachowywane w `ViewModel`, podczas zmian konfiguracji. Aby uzyskać dostęp `currentWordCount`do `GameFragment`programu , musisz udostępnić wersję tylko do odczytu przy użyciu właściwości kopii zapasowej.

1. W `GameViewModel`programie kliknij prawym przyciskiem myszy zmienną `currentWordCount`, wybierz **Refaktor** > **Zmień nazwę...** . Poprzedź nową nazwę podkreśleniem, `_currentWordCount`.
2. Dodaj pole zapasowe.

```
private var _currentWordCount = 0
val currentWordCount: Int
   get() = _currentWordCount
```

3. Wewnątrz , nad instrukcją return, dodaj kolejny dziennik, aby wydrukować dane aplikacji, słowo, wynik i liczbę słów `GameFragment`.`onCreateView()`

```
Log.d("GameFragment", "Word: ${viewModel.currentScrambledWord} " +
       "Score: ${viewModel.score} WordCount: ${viewModel.currentWordCount}")
```

4. W Android Studio otwórz **Logcat** , filtruj na `GameFragment`. Uruchom aplikację i zagraj w kilka słów. Zmień orientację swojego urządzenia. Fragment (kontroler interfejsu użytkownika) zostaje zniszczony i odtworzony. Obserwuj dzienniki. Teraz możesz zobaczyć, jak rośnie liczba punktów i słów!

```
com.example.android.unscramble D/GameFragment: GameFragment created/re-created!
com.example.android.unscramble D/GameFragment: GameViewModel created!
com.example.android.unscramble D/GameFragment: Word: oimfnru Score: 0 WordCount: 1
com.example.android.unscramble D/GameFragment: GameFragment destroyed!
com.example.android.unscramble D/GameFragment: GameFragment created/re-created!
com.example.android.unscramble D/GameFragment: Word: ofx Score: 80 WordCount: 5
com.example.android.unscramble D/GameFragment: GameFragment destroyed!
com.example.android.unscramble D/GameFragment: GameFragment created/re-created!
com.example.android.unscramble D/GameFragment: Word: ofx Score: 80 WordCount: 5
com.example.android.unscramble D/GameFragment: GameFragment destroyed!
com.example.android.unscramble D/GameFragment: GameFragment created/re-created!
com.example.android.unscramble D/GameFragment: Word: nvoiil Score: 160 WordCount: 9
com.example.android.unscramble D/GameFragment: GameFragment destroyed!
com.example.android.unscramble D/GameFragment: GameFragment created/re-created!
com.example.android.unscramble D/GameFragment: Word: nvoiil Score: 160 WordCount: 9
```

Zwróć uwagę, że dane aplikacji są zachowywane ViewModelpodczas zmian orientacji. Zaktualizujesz wartość wyniku i liczbę słów w interfejsie użytkownika za pomocą LiveDatai powiązania danych w późniejszych ćwiczeniach z programowania.

12. Zaktualizuj logikę restartu gry

1. Uruchom aplikację ponownie, zagraj w grę przez wszystkie słowa. W **Gratulacje!** w oknie alertu kliknij przycisk **ODTWÓRZ PONOWNIE** . Aplikacja nie pozwoli Ci ponownie grać, ponieważ liczba słów osiągnęła teraz wartość *MAX_NO_OF_WORDS*. Musisz zresetować liczbę słów do 0, aby ponownie zagrać w grę od początku.
2. Aby zresetować dane aplikacji, GameViewModeldodaj metodę o nazwie reinitializeData(). Ustaw wynik i liczbę słów na 0. Wyczyść listę słów i getNextWord()metodę wywołania.

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

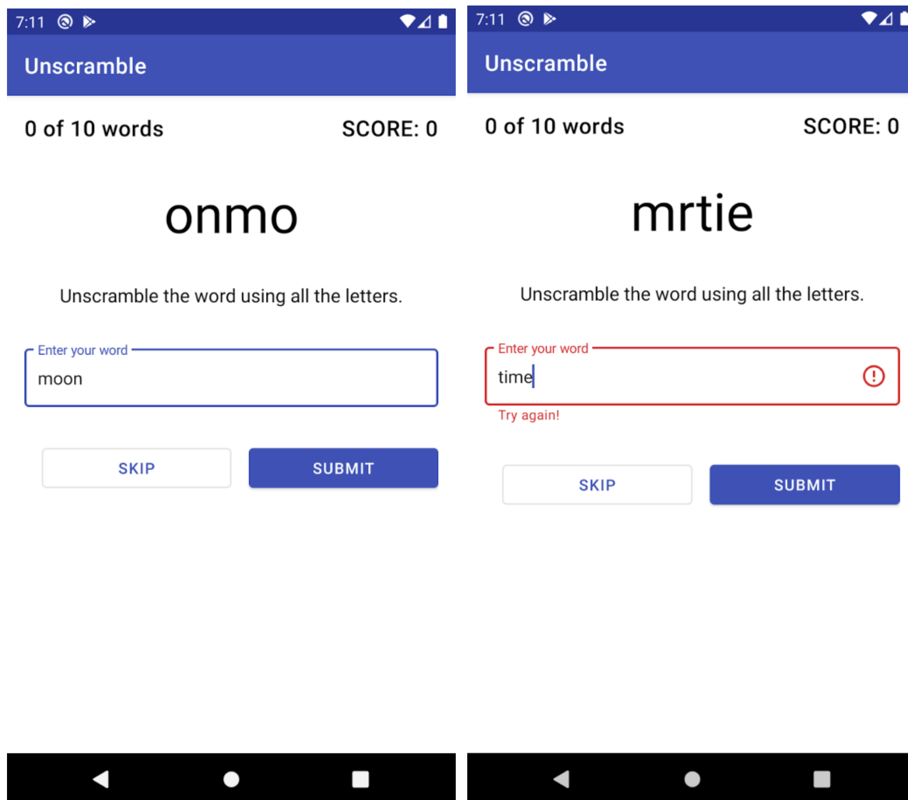
3. W GameFragmentgórnej części metody restartGame()wywołaj nowo utworzoną metodę reinitializeData().

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

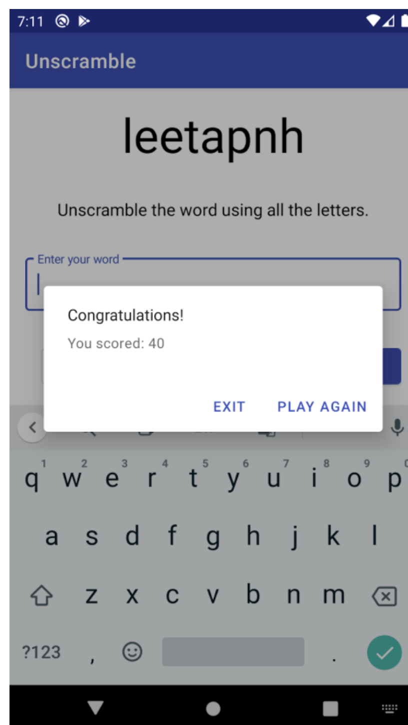
4. Uruchom ponownie swoją aplikację. Zagrać w grę. Gdy dojdziesz do okna z gratulacjami, kliknij **Odtwórz ponownie** . Teraz powinieneś być w stanie ponownie z powodzeniem zagrać w tę grę!

   Tak powinna wyglądać Twoja ostateczna aplikacja. Gra pokazuje dziesięć losowo zaszyfrowanych słów, które gracz może rozszyfrować. Możesz **pominąć** słowo lub odgadnąć słowo i stuknąć **Prześlij** . Jeśli zgadniesz poprawnie, wynik wzrasta. Nieprawidłowe odgadnięcie pokazuje stan błędu w polu tekstowym. Z każdym nowym słowem zwiększa się również liczba słów.

   Zauważ, że wynik i liczba słów wyświetlane na ekranie nie są jeszcze aktualizowane. Ale informacje są nadal przechowywane w modelu widoku i zachowywane podczas zmian konfiguracji, takich jak obracanie urządzenia. Zaktualizujesz wynik i liczbę słów na ekranie w późniejszych ćwiczeniach z programowania.



Pod koniec 10 słów gra się kończy i pojawia się okno dialogowe z ostrzeżeniem z końcowym wynikiem i opcją wyjścia z gry lub zagrania ponownie.



Gratulacje! Stworzyłeś swój pierwszy ViewModeli zapisałeś dane!

13. Kod rozwiązania

UWAGA : Upewnij się, że we wszystkich plikach źródłowych Kotlin zawsze umieszczasz nazwę pakietu swojej aplikacji.

GameFragment.kt

```
import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import com.example.android.unscramble.R
import com.example.android.unscramble.databinding.GameFragmentBinding
import com.google.android.material.dialog.MaterialAlertDialogBuilder

/**
 * Fragment where the game is played, contains the game logic.
 */
class GameFragment : Fragment() {

    private val viewModel: GameViewModel by viewModels()

    // Binding object instance with access to the views in the game_fragment.xml layout
    private lateinit var binding: GameFragmentBinding

    // Create a ViewModel the first time the fragment is created.
    // If the fragment is re-created, it receives the same GameViewModel instance created by the
    // first fragment

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        // Inflate the layout XML file and return a binding object instance
        binding = GameFragmentBinding.inflate(inflater, container, false)
        Log.d("GameFragment", "GameFragment created/re-created!")
        Log.d("GameFragment", "Word: ${viewModel.currentScrambledWord} " +
                "Score: ${viewModel.score} WordCount: ${viewModel.currentWordCount}")
        return binding.root
    }

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

/*
* Checks the user's word, and updates the score accordingly.
* Displays the next scrambled word.
* After the last word, the user is shown a Dialog with the final score.
*/
private fun onSubmitWord() {
    val playerWord = binding.textInputEditText.text.toString()

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

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

/*
 * Gets a random word for the list of words and shuffles the letters in it.
 */
private fun getNextScrambledWord(): String {
```

```
    val tempWord = allWordsList.random().toCharArray()
    tempWord.shuffle()
    return String(tempWord)
}

/*
* 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()
}

/*
 * Re-initializes the data in the ViewModel and updates the views with the new data, to
 * restart the game.
 */
private fun restartGame() {
    viewModel.reinitializeData()
    setErrorTextField(false)
    updateNextWordOnScreen()
}

/*
 * Exits the game.
 */
private fun exitGame() {
    activity?.finish()
}

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

/*
* Sets and resets the text field error status.
```

```
    */
    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
        }
    }

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

GameViewModel.kt

```
import android.util.Log
import androidx.lifecycle.ViewModel

/**
 * ViewModel containing the app data and methods to process the data
 */
class GameViewModel : ViewModel(){
    private var _score = 0
    val score: Int
        get() = _score

    private var _currentWordCount = 0
    val currentWordCount: Int
        get() = _currentWordCount

    private lateinit var _currentScrambledWord: String
    val currentScrambledWord: String
        get() = _currentScrambledWord

    // List of words used in the game
    private var wordsList: MutableList<String> = mutableListOf()
    private lateinit var currentWord: String

    init {
        Log.d("GameFragment", "GameViewModel created!")
```

```
    getNextWord()
}

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

/*
* 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)
    }
}

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


/*
* Increases the game score if the player's word is correct.
*/
private fun increaseScore() {
    _score += SCORE_INCREASE
}
```

```
    /*
    * Returns true if the player word is correct.
    * Increases the score accordingly.
    */
    fun isUserWordCorrect(playerWord: String): Boolean {
        if (playerWord.equals(currentWord, true)) {
            increaseScore()
            return true
        }
        return false
    }

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

14. Podsumowanie

- Wytyczne dotyczące architektury aplikacji na Androida zalecają oddzielenie klas, które mają różne obowiązki i sterowanie interfejsem użytkownika z modelu.
- Kontroler interfejsu użytkownika to klasa oparta na interfejsie użytkownika, taka jak `Activity`lub `Fragment`. Kontrolery interfejsu użytkownika powinny zawierać tylko logikę, która obsługuje interakcje interfejsu użytkownika i systemu operacyjnego; nie powinny być źródłem danych do wyświetlenia w interfejsie użytkownika. Umieść te dane i powiązaną logikę w `ViewModel`.
- Klasa `ViewModel`przechowuje i zarządza danymi związanymi z interfejsem użytkownika. Klasa `ViewModel`umożliwia przetrwanie danych przez zmiany konfiguracji, takie jak obracanie ekranu.
- `ViewModel`jest jednym z zalecanych składników architektury Androida.

15. Dowiedz się więcej

- ZobaczPrzegląd Modelu
- Przewodnik po architekturze aplikacji
- Praktyczne korzystanie z komponentów materiałowych dla systemu Android: dialogi
- Anatomia okna dialogowego alertu
- MaterialAlertDialogBuilder
- Właściwości podkładu
- Komponenty architektury Androida

- Okna dialogowe materiałów na Androida
- Właściwości i pola: gettery, settery, const, lateinit

Użyj LiveData z ViewModel

1. Zanim zaczniesz

Nauczyłeś się z poprzednich ćwiczeń z programowania, jak używać a ViewModeldo przechowywania danych aplikacji. ViewModelumożliwia przetrwanie danych aplikacji przez zmiany konfiguracji. W tym ćwiczeniu z programowania nauczysz się integrować LiveDataz danymi w ViewModel.
Klasa LiveDatajest również częścią składników architektury systemu Android i jest klasą uchwytu danych, którą można zaobserwować.

Warunki wstępne

- Jak pobrać kod źródłowy z GitHub i otworzyć go w Android Studio.
- Jak stworzyć i uruchomić podstawową aplikację na Androida w Kotlinie, korzystając z działań i fragmentów.
- Jak działają cykle życia aktywności i fragmentów.
- Jak zachować dane interfejsu użytkownika poprzez zmiany konfiguracji urządzenia przy użyciu ViewModel.
- Jak pisać wyrażenia lambda.

Czego się nauczysz

- Jak korzystać LiveDatai MutableLiveDataw swojej aplikacji.
- Jak hermetyzować dane przechowywane w a ViewModelza pomocą LiveData.
- Jak dodać metody obserwatora, aby obserwować zmiany wLiveData.
- Jak pisać wyrażenia powiązania w pliku układu.

Co zbudujesz

- Użyj LiveDatadla danych aplikacji (słowo, liczba słów i wynik) w aplikacji Rozszyfruj .
- Dodaj metody obserwatora, które są powiadamiane o zmianach danych, automatycznie aktualizuj widok tekstu zaszyfrowanego słowa.
- Zapisuj wyrażenia powiązania w pliku układu, które są wyzwalane, gdy LiveDatazmienia się element bazowy. Widoki punktacji, liczby słów i zaszyfrowanego tekstu są aktualizowane automatycznie.

Czego potrzebujesz

- Komputer z zainstalowanym Android Studio.
- Kod rozwiązania z poprzedniego ćwiczenia z programowania (Rozszyfruj aplikację za pomocą ViewModel).

Pobierz kod startowy do tego ćwiczenia z programowania

To laboratorium kodowania wykorzystuje aplikację Unscramble zbudowaną w poprzednim laboratorium (Przechowuj dane w ViewModel) jako kod startowy.
URL kodu startowego: https://github.com/google-developer-training/android-basics-kotlin-unscramble-app/tree/starter

Dodaj kod rozwiązania z poprzedniego ćwiczenia z programowania (Przechowuj dane w ViewModel) do powyższej gałęzi początkowej i użyj go jako kodu początkowego dla tego ćwiczenia z programowania.

2. Przegląd aplikacji startowej

W tym ćwiczeniu z programowania użyto kodu rozwiązania Rozszyfrowywanie, który znasz z poprzedniego ćwiczenia z programowania. Aplikacja wyświetla zaszyfrowane słowo, aby gracz mógł je odszyfrować. Gracz może próbować odgadnąć właściwe słowo dowolną ilość razy. Dane aplikacji, takie jak aktualne słowo, wynik gracza i liczba słów, są zapisywane w ViewModel. Jednak interfejs użytkownika aplikacji nie odzwierciedla nowych wartości wyniku i liczby słów. W tym ćwiczeniu z programowania zaimplementujesz brakujące funkcje za pomocą LiveData.

3. Co to są dane na żywo

LiveDatato obserwowalna klasa posiadacza danych, która jest świadoma cyklu życia.

Niektóre cechy LiveData:

- LiveDataprzechowuje dane; LiveDatato opakowanie, którego można używać z dowolnym typem danych.
- LiveDatajest obserwowalny, co oznacza, że obserwator jest powiadamiany o LiveDatazmianie danych posiadanych przez obiekt.
- LiveDatajest świadomy cyklu życia. Kiedy dołączasz obserwatora do LiveData, obserwator jest kojarzony z LifecycleOwner(zwykle z czynnością lub fragmentem). Jedyne LiveDataobserwatory aktualizacji, które są w aktywnym stanie cyklu życia, takie jak STARTEDlub RESUMED. Więcej informacji LiveDatai obserwacji można przeczytać tutaj .

Aktualizacja interfejsu użytkownika w kodzie startowym

W kodzie startowym updateNextWordOnScreen()metoda jest wywoływana jawnie, za każdym razem, gdy chcesz wyświetlić nowe zaszyfrowane słowo w interfejsie użytkownika. Ta metoda jest wywoływana podczas inicjowania gry i gdy gracze naciskają przycisk **Prześlij** lub **Pomiń** . Ta metoda jest wywoływana z
metod onViewCreated(), restartGame(), onSkipWord()i onSubmitWord(). Dzięki Livedata, nie będziesz musiał wywoływać tej metody z wielu miejsc, aby zaktualizować interfejs użytkownika. Zrobisz to tylko raz w obserwatorze.

4. Dodaj LiveData do bieżącego zaszyfrowanego słowa

W tym zadaniu dowiesz się, jak zawijać dowolne dane LiveData,, konwertując bieżące słowo GameViewModelw [LiveData](). W kolejnym zadaniu dodasz obserwatora do tych LiveDataobiektów i nauczysz się obserwować LiveData.

MutableLiveData

MutableLiveDatajest zmienną wersją LiveData, to znaczy, że wartość przechowywanych w niej danych może zostać zmieniona.

1. W GameViewModelprogramie zmień typ zmiennej _currentScrambledWordna . i są klasami generycznymi, więc musisz określić typ danych, które przechowują.[MutableLiveData]()<String>LiveDataMutableLiveData
2. Zmień typ zmiennej na _currentScrambledWordna, valponieważ wartość obiektu LiveData/ MutableLiveDatapozostanie taka sama, a zmienią się tylko dane przechowywane w obiekcie.

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

3. Zmień pole zapasowe, currentScrambledWordwpisz na LiveData<String>, ponieważ jest niezmienne. Android Studio pokaże kilka błędów, które naprawisz w kolejnych krokach.

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

4. Aby uzyskać dostęp do danych w LiveDataobiekcie, użyj valuewłaściwości. Wewnątrz GameViewModelmetody getNextWord(), w elsebloku, zmień odniesienie _currentScrambledWorddo _currentScrambledWord.value.

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

5. Dołącz obserwatora do obiektu LiveData

W tym zadaniu konfigurujesz obserwatora w komponencie aplikacji, GameFragment. Dodany obserwator obserwuje zmiany w danych aplikacji currentScrambledWord. LiveDatajest świadomy cyklu życia, co oznacza, że aktualizuje tylko obserwatorów, którzy są w aktywnym stanie cyklu życia. Tak więc obserwator w GameFragmenttestamencie zostanie powiadomiony tylko wtedy, gdy GameFragmentjest w STARTEDlub RESUMEDstany.

1. W GameFragmentprogramie usuń metodę updateNextWordOnScreen()i wszystkie jej wywołania. Nie potrzebujesz tej metody, ponieważ będziesz dołączać obserwatora do LiveData.
2. W onSubmitWord()programie zmodyfikuj pusty if-elseblok w następujący sposób. Kompletna metoda powinna wyglądać tak.

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

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

3. Dołącz obserwatora dla currentScrambledWord LiveData. Na GameFragmentkońcu wywołania zwrotnego onViewCreated()wywołaj observe()metodę on currentScrambledWord.

```
// Observe the currentScrambledWord LiveData.
viewModel.currentScrambledWord.observe()
```

Android Studio wyświetli błąd dotyczący brakujących parametrów. Naprawisz błąd w następnym kroku.

4. Przekaż *viewLifecycleOwner*jako pierwszy parametr do observe()metody. *viewLifecycleOwner*reprezentuje cykl życia widoku fragmentu . Ten parametr pomaga LiveDatabyć świadomym GameFragmentcyklu życia i powiadamiać obserwatora tylko wtedy, gdy GameFragmentjest w stanie aktywnym (STARTEDlub RESUMED).
5. Dodaj lambdę jako drugi parametr z newWordparametrem funkcji. newWordBędzie zawierał nową zaszyfrowaną wartość słowa .

```
// Observe the scrambledCharArray LiveData, passing in the LifecycleOwner and the observer.
viewModel.currentScrambledWord.observe(viewLifecycleOwner,
  { newWord ->
  })
```

Wyrażenie lambda to funkcja anonimowa, która nie jest zadeklarowana, ale jest przekazywana natychmiast jako wyrażenie. Wyrażenie lambda jest zawsze otoczone nawiasami klamrowymi { }.

6. W treści funkcji wyrażenia lambda przypisz newWorddo widoku tekstu zaszyfrowanego słowa.

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

7. Skompiluj i uruchom aplikację. Twoja aplikacja do gry powinna działać dokładnie tak samo, jak poprzednio, ale teraz widok tekstu zaszyfrowanego słowa jest automatycznie aktualizowany w `LiveData`obserwatorze, a nie w `updateNextWordOnScreen()`metodzie.

6. Dołącz obserwatora do punktacji i liczby słów

Podobnie jak w poprzednim zadaniu, w tym zadaniu dodasz `LiveData`do innych danych w aplikacji wynik i liczbę słów, dzięki czemu interfejs użytkownika będzie aktualizowany poprawnymi wartościami wyniku i liczby słów podczas gry.

Krok 1: Zawijaj wynik i liczbę słów za pomocą LiveData

1. W `GameViewModel`programie zmień typ zmiennych `_score`i `_currentWordCount`class na `val`.
2. Zmień typ danych zmiennych `_score`i `_currentWordCount` na `MutableLiveData`i zainicjuj je na `0`.
3. Zmień typ pól zapasowych na`LiveData<Int>`.

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

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

4. Na `GameViewModel`początku `reinitializeData()`metody zmień odwołanie odpowiednio do `_score`i `_currentWordCount` do `_score.value`i `_currentWordCount.value`.

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

5. W `GameViewModel`, wewnątrz `nextWord()`metody, zmień odwołanie `_currentWordCount` do `_currentWordCount.value!!`.

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

```
    } else false
  }
```

6. W GameViewModel, wewnątrz metod increaseScore()i getNextWord()zmień odpowiednio odwołanie do _scorei _currentWordCount do _score.*value*i _currentWordCount.*value*. Android Studio wyświetli błąd, ponieważ _scorenie jest już liczbą całkowitą, to LiveData, naprawisz to w następnych krokach.
7. Użyj [plus()]funkcji Kotlin, aby zwiększyć _scorewartość, która wykonuje dodawanie z wartością null-safety.

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

8. Podobnie użyj [inc()]funkcji Kotlin, aby zwiększyć wartość o jeden z wartością null-safety.

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

9. W GameFragmentprogramie uzyskaj dostęp do wartości scorekorzystania z valuewłaściwości. Wewnątrz showFinalScoreDialog()metody zmień viewModel.scorena viewModel.score.*value*.

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

Krok 2: Dołącz obserwatorów do punktacji i liczby słów

W aplikacji wynik i liczba słów nie są aktualizowane. Zaktualizujesz je w tym zadaniu za pomocą LiveDataobserwatorów.

1. Wewnątrz GameFragmentmetody onViewCreated()usuń kod, który aktualizuje widoki tekstowe wyników i liczby słów.

   Usunąć:

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

2. Na GameFragmentkońcu onViewCreated()metody dołącz obserwatora dla score. Przekaż viewLifecycleOwnerjako pierwszy parametr obserwatorowi i wyrażenie lambda dla drugiego parametru. Wewnątrz wyrażenia lambda przekaż nowy wynik jako parametr, a wewnątrz treści funkcji ustaw nowy wynik w widoku tekstowym.

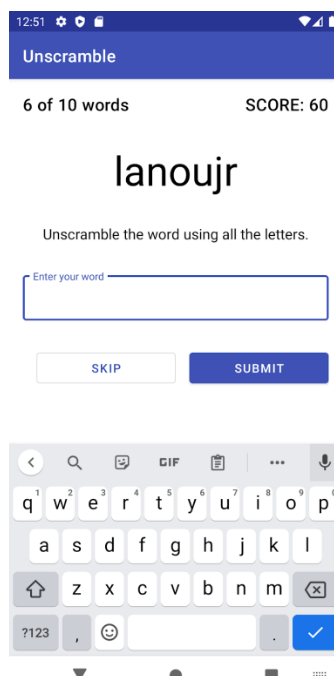
```
viewModel.score.observe(viewLifecycleOwner,
  { newScore ->
     binding.score.text = getString(R.string.score, newScore)
  })
```

3. Na końcu onViewCreated()metody dołącz obserwatora dla currentWordCount LiveData. Przekaż viewLifecycleOwnerjako pierwszy parametr obserwatorowi i wyrażenie lambda dla drugiego parametru. Wewnątrz wyrażenia lambda przekaż nową liczbę słów jako parametr, aw treści funkcji ustaw nową liczbę słów wraz z MAX_NO_OF_WORDSwidokiem tekstowym.

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

   Nowi obserwatorzy zostaną wyzwoleni, gdy wartość wyniku i liczba słów zmieni się w ciągu ViewModel, w okresie życia właściciela cyklu życia, czyli w GameFragment.

4. Uruchom swoją aplikację, aby zobaczyć magię. Zagraj w grę za pomocą kilku słów. Wynik i liczba słów są również poprawnie aktualizowane na ekranie. Zauważ, że nie aktualizujesz tych widoków tekstowych na podstawie pewnych warunków w kodzie. I są scorei odpowiadające im obserwatorzy są wywoływane automatycznie, gdy zmienia się podstawowa wartość.currentWordCountLiveData



7. Użyj LiveData z wiązaniem danych

W poprzednich zadaniach Twoja aplikacja nasłuchuje zmian danych w kodzie. Podobnie aplikacje mogą nasłuchiwać zmian danych z układu. W przypadku powiązania danych, gdy obserwowalna `LiveData`wartość ulegnie zmianie, elementy interfejsu użytkownika w układzie, z którym jest powiązany, są również powiadamiane, a interfejs użytkownika można zaktualizować z poziomu układu.

Koncepcja: Wiązanie danych

W poprzednich ćwiczeniach z programowania widziałeś powiązanie [widoku](#) , które jest wiązaniem jednokierunkowym. Widoki można powiązać z kodem, ale nie odwrotnie.

Odświeżacz dla powiązania widoku:

Wiązanie widoków to funkcja, która umożliwia łatwiejszy dostęp do widoków w kodzie. Generuje klasę powiązania dla każdego pliku układu XML. Wystąpienie klasy powiązania zawiera bezpośrednie odwołania do wszystkich widoków, które mają identyfikator w odpowiednim układzie. Na przykład aplikacja Unscramble obecnie używa powiązania widoku, więc do widoków można się odwoływać w kodzie przy użyciu wygenerowanej klasy powiązania.

Przykład:

```
binding.textViewUnscrambledWord.text = newWord
binding.score.text = getString(R.string.score, newScore)
binding.wordCount.text =
        getString(R.string.word_count, newWordCount, MAX_NO_OF_WORDS)
```

Za pomocą powiązania widoku nie można odwoływać się do danych aplikacji w widokach (plikach układu). Można to osiągnąć za pomocą [powiązania danych](#) .

Wiązanie danych

Data Binding Library jest również częścią [biblioteki Android Jetpack](#) . Powiązanie danych wiąże składniki interfejsu użytkownika w układach ze źródłami danych w aplikacji przy użyciu formatu deklaratywnego, którego nauczysz się w dalszej części ćwiczenia z programowania.

W uproszczeniu Wiązanie danych to powiązanie danych (z kodu) z widokami + powiązanie widoków (powiązanie widoków z kodem)

Przykład użycia powiązania widoku w kontrolerze UI

```
binding.textViewUnscrambledWord.text = viewModel.currentScrambledWord
```

Przykład użycia wiązania danych w pliku układu

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

Powyższy przykład pokazuje, jak używać biblioteki powiązań danych do przypisywania danych aplikacji do widoków/widżetów bezpośrednio w pliku układu. Zwróć uwagę na użycie `@{}`składni w wyrażeniu przypisania.

Główną zaletą korzystania z powiązania danych jest to, że pozwala usunąć wiele wywołań frameworka UI z twoich działań, co czyni je prostszymi i łatwiejszymi w utrzymaniu. Może to również poprawić wydajność aplikacji i pomóc w zapobieganiu wyciekom pamięci i wyjątkom zerowego wskaźnika.

Krok 1: Zmień powiązanie widoku na powiązanie danych

1. W `build.gradle(Module)`pliku włącz `dataBinding`właściwość w `buildFeatures`sekcji.

   Zastępować

   ```
   buildFeatures {
     viewBinding = true
   }
   ```

   z

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

   Wykonaj synchronizację Gradle po wyświetleniu monitu przez Android Studio.

2. Aby użyć wiązania danych w dowolnym projekcie Kotlin, należy zastosować `kotlin-kapt`wtyczkę. Ten krok jest już wykonany za Ciebie w `build.gradle(Module)`pliku.

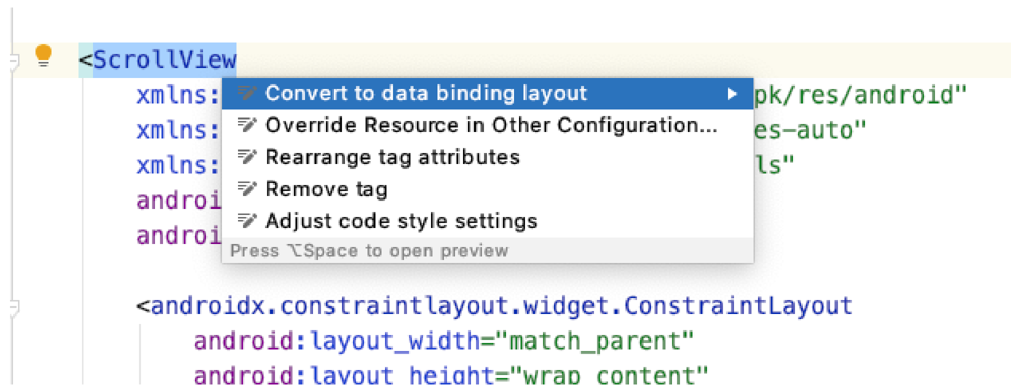
   ```
   plugins {
     id 'com.android.application'
     id 'kotlin-android'
     id 'kotlin-kapt'
   }
   ```

   Powyższe kroki automatycznie generują klasę powiązania dla każdego pliku XML układu w aplikacji. Jeśli nazwa pliku układu to `activity_main.xml`wtedy twoja klasa autogen zostanie wywołana `ActivityMainBinding`.

Krok 2: Konwertuj plik układu na układ powiązania danych

Pliki układu powiązania danych są nieco inne i zaczynają się od tagu głównego, `<layout>`po którym następuje element opcjonalny `<data>`i element `view`główny. Ten element widoku jest tym, czym byłby twój katalog główny w niewiążącym pliku układu.

1. Otwórz `game_fragment.xml`, wybierz kartę **kodu .**
2. Aby przekonwertować układ na układ powiązania danych, zawiń element główny w `<layout>`znacznik. Będziesz także musiał przenieść definicje przestrzeni nazw (atrybuty zaczynające się od `xmlns:`) do nowego elementu głównego. Dodaj `<data></data>`tagi wewnątrz `<layout>`tagu nad elementem głównym. Android Studio oferuje wygodny sposób na zrobienie tego automatycznie: kliknij prawym przyciskiem myszy element główny (`ScrollView`), wybierz **Pokaż akcje kontekstowe** > **Konwertuj na układ powiązania danych**



3. Twój układ powinien wyglądać mniej więcej tak:

```
<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>
```

4. W `GameFragment`, na początku `onCreateView()`metody, zmień wystąpienie `binding`zmiennej, aby użyć powiązania danych.

   Zastępować

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

   Z

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

5. Skompiluj kod; powinieneś być w stanie skompilować bez żadnych problemów. Twoja aplikacja używa teraz powiązania danych, a widoki w układzie mogą uzyskiwać dostęp do danych aplikacji.

8. Dodaj zmienne wiążące dane

W tym zadaniu dodasz właściwości w pliku układu, aby uzyskać dostęp do danych aplikacji z viewModel. Zainicjujesz zmienne układu w kodzie.

1. W game_fragment.xml, wewnątrz <data>znacznika dodaj znacznik podrzędny o nazwie <variable>, zadeklaruj właściwość o nazwie gameViewModeli typu GameViewModel. Użyjesz tego do powiązania danych ViewModelz układem.

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

Zwróć uwagę, że typ gameViewModelzawiera nazwę pakietu. Upewnij się, że ta nazwa pakietu jest zgodna z nazwą pakietu w Twojej aplikacji.

2. Pod gameViewModeldeklaracją dodaj kolejną zmienną wewnątrz <data>tagu type Integeri nazwij ją maxNoOfWords. Użyjesz tego do powiązania ze zmienną w ViewModel, aby przechowywać liczbę słów na grę.

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

3. Na GameFragmentpoczątku onViewCreated()metody zainicjuj zmienne układu gameViewModeli maxNoOfWords.

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

   binding.gameViewModel = viewModel

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

4. Jest LiveDataobserwowalny z uwzględnieniem cyklu życia, więc musisz przekazać właściciela cyklu życia do układu. W GameFragment, wewnątrz onViewCreated()metody, poniżej inicjalizacji zmiennych powiązania, dodaj następujący kod.

```
   // 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
```

Przypomnij sobie, że zaimplementowałeś podobną funkcjonalność podczas implementacji LiveDataobserwatorów. Przekazałeś viewLifecycleOwnerjako jeden z parametrów LiveDataobserwatorom.

9. Użyj wyrażeń wiążących

Wyrażenia powiązania są zapisywane w układzie we właściwościach atrybutu (takich jak android:text), odwołując się do właściwości układu. Właściwości układu są deklarowane w górnej części pliku układu powiązania danych za pomocą <variable>znacznika. Gdy zmieni się dowolna zmienna zależna, 'Biblioteka DB' uruchomi wyrażenia wiążące (i w ten sposób zaktualizuje widoki). To wykrywanie zmian to świetna optymalizacja, którą otrzymujesz za darmo, gdy korzystasz z biblioteki powiązań danych.

Składnia wyrażeń wiążących

Wyrażenia wiążące zaczynają się od @symbolu i są owinięte w nawiasy klamrowe {}. W poniższym przykładzie TextViewtekst jest ustawiony na firstNamewłaściwość userzmiennej:

Przykład:

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

Krok 1: Dodaj wyrażenie wiążące do bieżącego słowa

W tym kroku powiążesz bieżący widok tekstu słowa z LiveDataobiektem w ViewModel.

1. W programie game_fragment.xmldodaj textatrybut do textView_unscrambled_wordwidoku tekstu. Użyj nowej zmiennej układu gameViewModeli przypisz @{gameViewModel.currentScrambledWord}do textatrybutu.

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

2. W GameFragmentprogramie usuń LiveDatakod obserwatora dla currentScrambledWord: Nie potrzebujesz już kodu obserwatora we fragmencie. Układ otrzymuje aktualizacje zmian LiveDatabezpośrednio.

Usunąć

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

3. Uruchom swoją aplikację, Twoja aplikacja powinna działać jak poprzednio. Ale teraz widok tekstu zaszyfrowanego słowa używa wyrażeń wiążących do aktualizacji interfejsu użytkownika, a nie LiveDataobserwatorów.

Krok 2: Dodaj wiążące wyrażenie do partytury i liczbę słów

Zasoby w wyrażeniach wiążących dane

Wyrażenie powiązania danych może odwoływać się do zasobów aplikacji przy użyciu następującej składni.

Przykład:

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

W powyższym przykładzie paddingatrybutowi przypisywana jest wartość largePaddingz dimen.xmlpliku zasobów.

Możesz również przekazać właściwości układu jako parametry zasobów.

Przykład:

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

strings.xml

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

W powyższym przykładzie example_resourcejest zasobem ciągu z %ssymbolem zastępczym. Przekazujesz user.lastNamejako parametr zasobu w wyrażeniu powiązania, gdzie userjest zmienną układu.

W tym kroku dodasz wyrażenia wiążące do widoków punktacji i liczby słów, przekazując parametry zasobów. Ten krok jest podobny do tego, co zrobiłeś textView_unscrambled_wordpowyżej.

1. W game_fragment.xmlprogramie zaktualizuj textatrybut dla word_countwidoku tekstu za pomocą następującego wyrażenia powiązania. Użyj word_countzasobu ciągu i przekaż w gameViewModel.currentWordCounti maxNoOfWordsjako parametry zasobu.

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

2. Zaktualizuj textatrybut dla scorewidoku tekstu za pomocą następującego wyrażenia powiązania. Użyj scorezasobu ciągu i przekaż gameViewModel.scorejako parametr zasobu.

```
<TextView
  android:id="@+id/score"
```

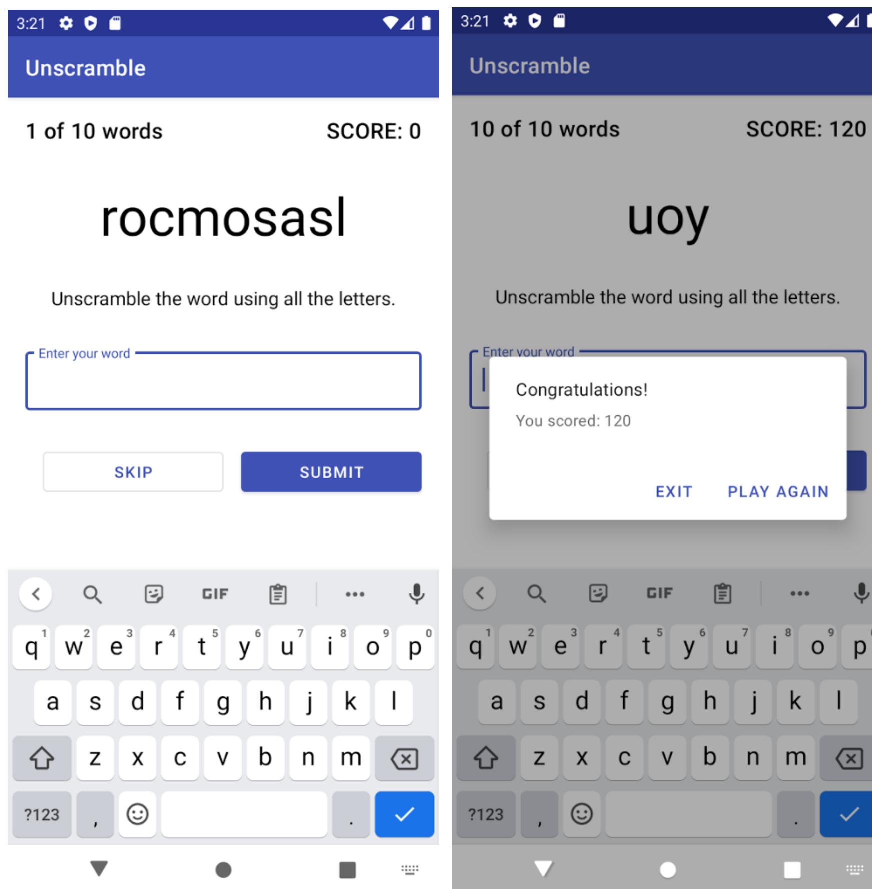
```
...
android:text="@{@string/score(gameViewModel.score)}"
... />
```

3. Usuń LiveDataobserwatorów z GameFragment. Nie potrzebujesz ich już, wyrażenia wiążące aktualizują interfejs użytkownika, gdy odpowiednie LiveDatazmiany.

   Usunąć:

```
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)
  })
```

4. Uruchom aplikację i zagraj w kilka słów. Teraz Twój kod używa LiveDatai powiązań wyrażeń, aby zaktualizować interfejs użytkownika.



Gratulacje! Nauczyłeś się używać LiveDataz LiveDataobserwatorami i LiveDatawyrażeniami wiążącymi.

10. Przetestuj aplikację Unscramble z włączoną funkcją Talkback

Jak poznałeś podczas tego kursu, chcesz tworzyć aplikacje dostępne dla jak największej liczby użytkowników. Niektórzy użytkownicy mogą używać Talkback do uzyskiwania dostępu do aplikacji i poruszania się po niej. TalkBack to czytnik ekranu Google dostępny na urządzeniach z Androidem. TalkBack zapewnia komunikaty głosowe, dzięki czemu możesz korzystać z urządzenia bez patrzenia na ekran.

Po włączeniu Talkback upewnij się, że gracz może grać w grę.

1. Włącz TalkBack na swoim urządzeniu, postępując zgodnie z tymi instrukcjami .
2. Wróć do aplikacji Rozszyfruj.
3. Poznaj swoją aplikację za pomocą Talkback, korzystając z tych instrukcji . Przesuń palcem w prawo, aby poruszać się po elementach ekranu w kolejności, i przesuń w lewo, aby przejść w przeciwnym kierunku. Kliknij dwukrotnie w dowolnym miejscu, aby wybrać. Sprawdź, czy możesz dotrzeć do wszystkich elementów aplikacji za pomocą gestów machnięcia.
4. Upewnij się, że użytkownik Talkback może przejść do każdego elementu na ekranie.
5. Zauważ, że Talkback próbuje odczytać zaszyfrowane słowo jako słowo. Może to być mylące dla gracza, ponieważ nie jest to prawdziwe słowo.
6. Lepszym doświadczeniem użytkownika byłoby, gdyby Talkback odczytał na głos poszczególne znaki zaszyfrowanego słowa. W obrębie GameViewModelskonwertuj zaszyfrowane słowo Stringna Spannableciąg. Spanable string to string z dołączonymi do niego dodatkowymi informacjami. W tym przypadku chcemy powiązać ciąg z TtsSpanof , aby mechanizm zamiany tekstu na mowę odczytał na głos zaszyfrowane słowo dosłownie, znak po znaku.TYPE_VERBATIM,
7. W GameViewModel,celu zmodyfikowania sposobu currentScrambledWorddeklarowania zmiennej użyj następującego kodu:

```
val currentScrambledWord: LiveData<Spannable> = Transformations.map(_currentScrambledWord) {
    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
    }
}
```

Ta zmienna jest teraz LiveData<Spannable>zamiast LiveData<String>. Nie musisz się martwić o zrozumienie wszystkich szczegółów tego, jak to działa, ale implementacja używa LiveDatatransformacji do przekonwertowania bieżącego zaszyfrowanego słowa Stringna ciąg Spannable, który może być odpowiednio obsłużony przez usługę ułatwień dostępu. W następnym ćwiczeniu z programowania dowiesz się więcej o LiveDataprzekształceniach, które pozwalają zwrócić inną LiveDatainstancję na podstawie wartości odpowiadającej LiveData.

8. Uruchom aplikację Unscramble, poznaj swoją aplikację za pomocą Talkback. TalkBack powinien teraz odczytać poszczególne znaki zaszyfrowanego słowa.

   Aby uzyskać więcej informacji o tym, jak zwiększyć dostępność aplikacji, zapoznaj się z tymi zasadami .
   Uwaga : Usługa ułatwień dostępu dostępna na Twoim urządzeniu może się różnić w zależności od producenta urządzenia, możesz doświadczyć różnych zachowań. Jeśli poszczególne znaki nie są odczytywane na głos dla zaszyfrowanego słowa, spróbuj uruchomić aplikację na emulatorze w Android Studio. Aby włączyć Talkback , musisz zainstalować aplikację Android Accessibility Suite na emulatorze.

11. Usuń nieużywany kod

Dobrą praktyką jest usuwanie martwego, nieużywanego, niechcianego kodu dla kodu rozwiązania. Dzięki temu kod jest łatwy w utrzymaniu, co ułatwia również nowym członkom zespołu lepsze zrozumienie kodu.

1. W GameFragmentprogramie usuń getNextScrambledWord()i onDetach()metody.
2. W metodzie GameViewModelusuwania .onCleared()
3. Usuń wszystkie nieużywane importy w górnej części plików źródłowych. Będą wyszarzone.

   Nie potrzebujesz już instrukcji dziennika, możesz je usunąć z kodu, jeśli wolisz.

1. [Opcjonalnie] Usuń Loginstrukcje w plikach źródłowych (GameFragment.kti GameViewModel.kt), które dodałeś w poprzednim ćwiczeniu z programowania, aby zrozumieć ViewModelcykl życia.

12. Kod rozwiązania

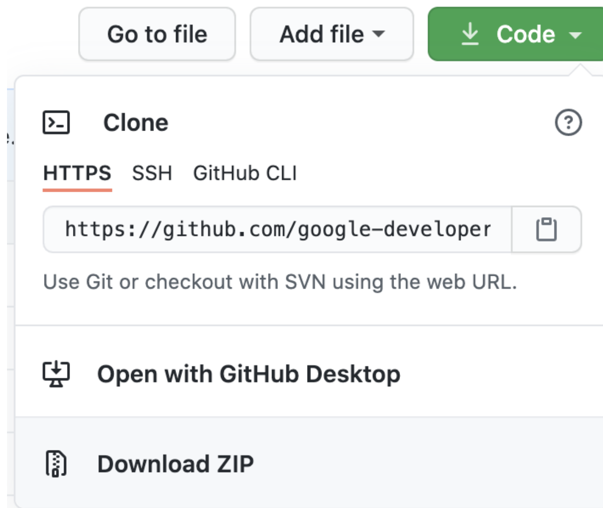
Kod rozwiązania dla tego ćwiczenia programowania znajduje się w projekcie pokazanym poniżej.

Adres URL kodu rozwiązania: https://github.com/google-developer-training/android-basics-kotlin-unscramble-app/tree/main

Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

Pobierz kod

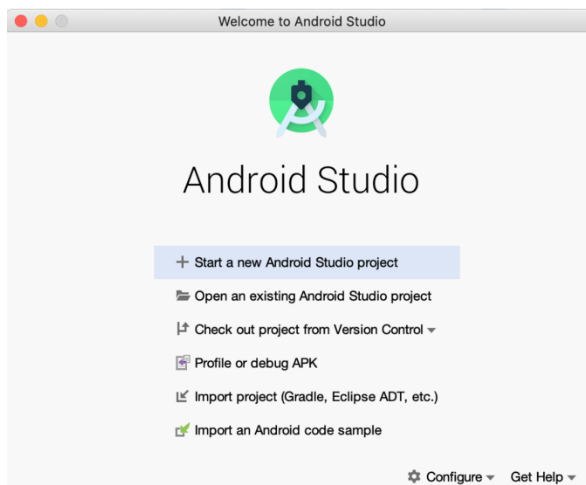
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



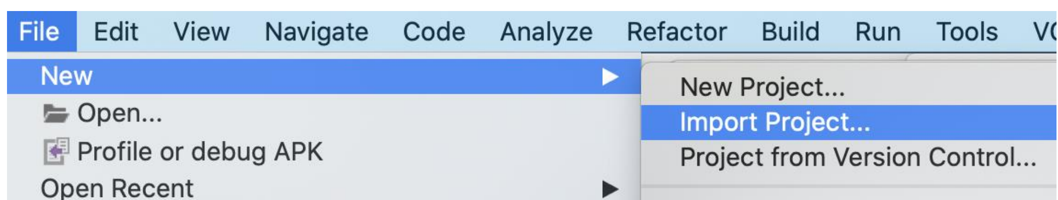
3. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
4. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
5. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

Otwórz projekt w Android Studio

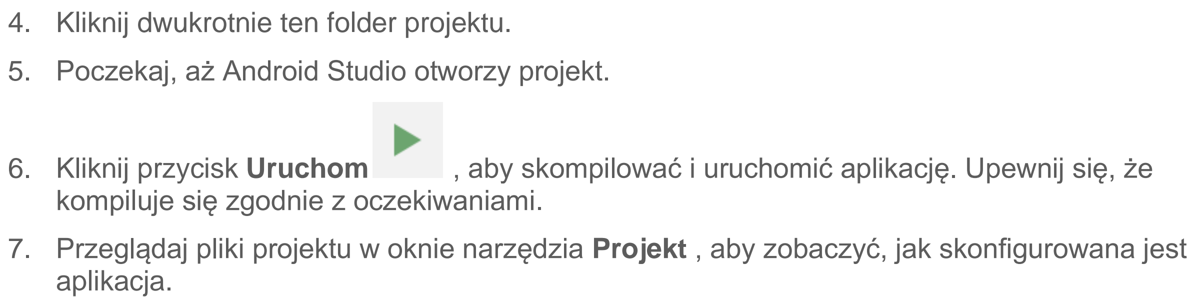
1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz istniejący projekt Android Studio.**



Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Nowy** > **Importuj projekt** .



3. W oknie dialogowym **Importuj projekt** przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).

4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.
6. Kliknij przycisk **Uruchom**  , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.
7. Przeglądaj pliki projektu w oknie narzędzia **Projekt** , aby zobaczyć, jak skonfigurowana jest aplikacja.

13. Podsumowanie

- LiveDataprzechowuje dane; LiveDatato wrapper, którego można używać z dowolnymi danymi
- LiveDatajest obserwowalny, co oznacza, że obserwator jest powiadamiany o LiveDatazmianie danych posiadanych przez obiekt.
- LiveDatajest świadomy cyklu życia. Po dołączeniu obserwatora do LiveData, obserwator jest skojarzony z LifecycleOwner(zwykle działaniem lub fragmentem). LiveData aktualizuje tylko obserwatorów, którzy są w aktywnym stanie cyklu życia, takich jak STARTEDlub RESUMED. Więcej informacji LiveDatai obserwacji można przeczytać tutaj .
- Aplikacje mogą nasłuchiwać zmian LiveData z układu przy użyciu powiązania danych i wyrażeń powiązań.
- Wyrażenia powiązania są zapisywane w układzie we właściwościach atrybutu (takich jak android:text), odwołując się do właściwości układu.

14. Dowiedz się więcej

- Przegląd danych na żywo
- Dokumentacja API obserwatora LiveData
- Wiązanie danych
- Dwukierunkowe wiązanie danych
- Korzystanie z biblioteki powiązań danych

Posty na blogu

- Wiązanie danych — wyciągnięte wnioski. Biblioteka powiązań danych (odnosi się do... | autorstwa Chrisa Banesa | programiści Androida)

Przykłady zaawansowanych aplikacji nawigacyjnych

Połącz wszystko, czego nauczyłeś się w tej części o nawigacji, ViewModel, wiązaniu danych i LiveData, tworząc bardziej zaawansowaną aplikację, która obejmuje również niestandardowe zachowanie stosu wstecznego.

Wspólny model widoku we fragmentach

1. Zanim zaczniesz

Nauczyłeś się korzystać z działań, fragmentów, intencji, wiązania danych, komponentów nawigacyjnych i podstaw komponentów architektury. W tym ćwiczeniu z kodowania złożysz wszystko razem i będziesz pracować nad zaawansowaną próbką, aplikacją do zamawiania babeczek.

Dowiesz się, jak korzystać z udostępnionego ViewModel do udostępniania danych między fragmentami tej samej czynności i nowymi koncepcjami, takimi jak LiveData przekształcenia.

Warunki wstępne

- Wygodne czytanie i rozumienie układów Androida w XML
- Zapoznanie się z podstawami komponentu nawigacyjnego Jetpack
- Potrafi stworzyć wykres nawigacyjny z miejscami docelowymi fragmentów w aplikacji
- Czy wcześniej używałeś fragmentów w ramach działania
- Może utworzyć ViewModel do przechowywania danych aplikacji
- Może używać powiązania danych z LiveData, aby zachować aktualność interfejsu użytkownika z danymi aplikacji w ViewModel

Czego się nauczysz

- Jak wdrożyć zalecane praktyki architektury aplikacji w bardziej zaawansowanym przypadku użycia
- Jak korzystać ze współdzielonego ViewModel fragmentu w działaniu
- Jak zastosować LiveData transformację

Co zbudujesz

- Aplikacja **Cupcake** , która wyświetla przepływ zamówień na babeczki, umożliwiając użytkownikowi wybór smaku babeczki, ilości i daty odbioru.

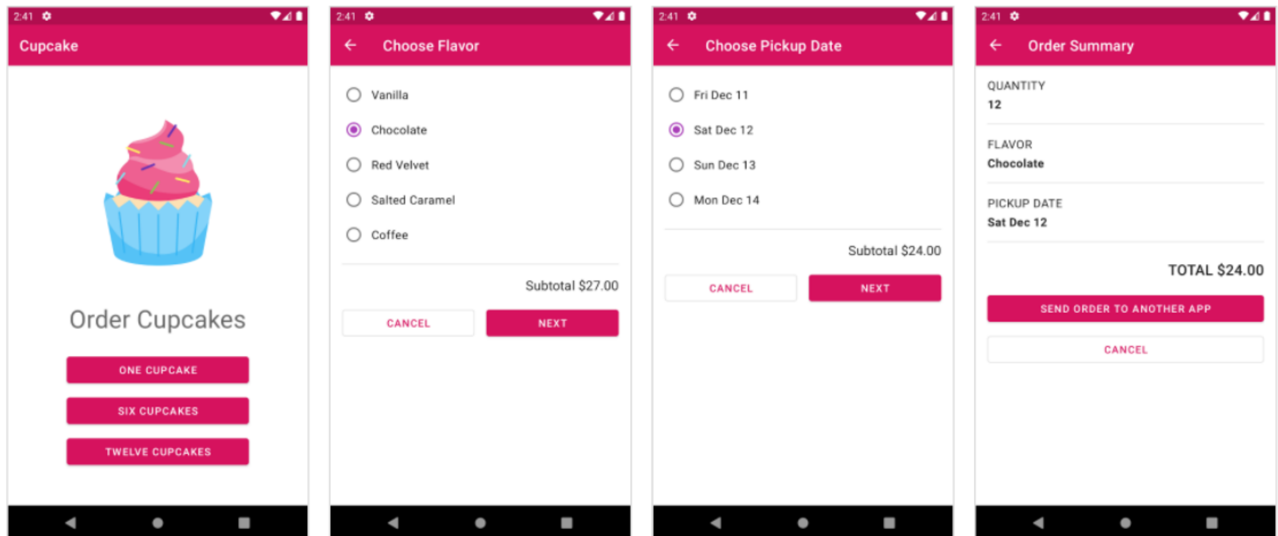
Czego potrzebujesz

- Komputer z zainstalowanym Android Studio.
- Kod startowy aplikacji **Cupcake** .

2. Przegląd aplikacji startowej

Przegląd aplikacji Cupcake

Aplikacja Cupcake pokazuje, jak zaprojektować i wdrożyć aplikację do zamawiania online. Na końcu tej ścieżki ukończysz aplikację **Cupcake** z następującymi ekranami. Użytkownik może wybrać ilość, smak i inne opcje zamówienia babeczek.



Pobierz kod startowy do tego ćwiczenia z programowania

To ćwiczenie z programowania zapewnia kod startowy, który można rozszerzyć o funkcje nauczane w tym ćwiczeniu z programowania. Kod startowy będzie zawierał kod, który jest Ci znany z poprzednich ćwiczeń z programowania.

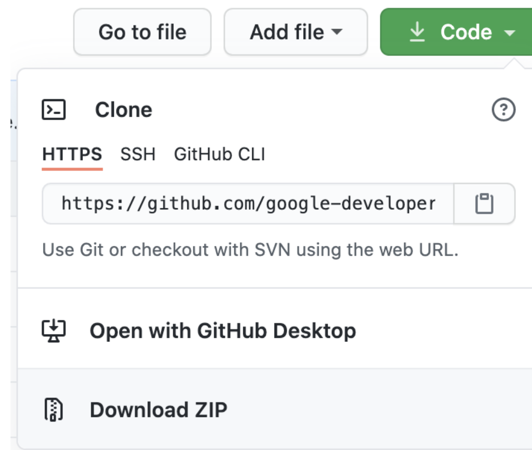
Jeśli pobierzesz kod startowy z GitHub, zwróć uwagę, że nazwa folderu projektu to `android-basics-kotlin-cupcake-app-starter`. Wybierz ten folder podczas otwierania projektu w Android Studio.

URL kodu startowego: https://github.com/google-developer-training/android-basics-kotlin-cupcake-app/tree/starter

Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

Pobierz kod

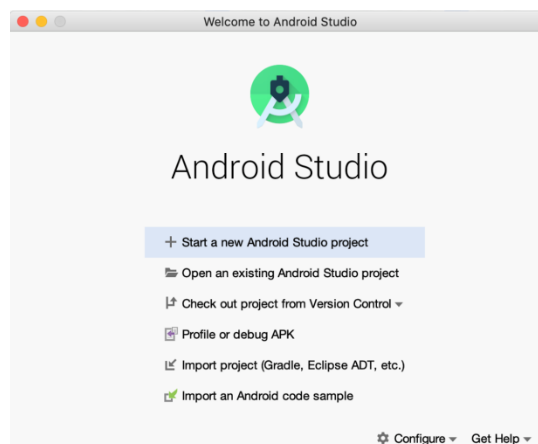
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



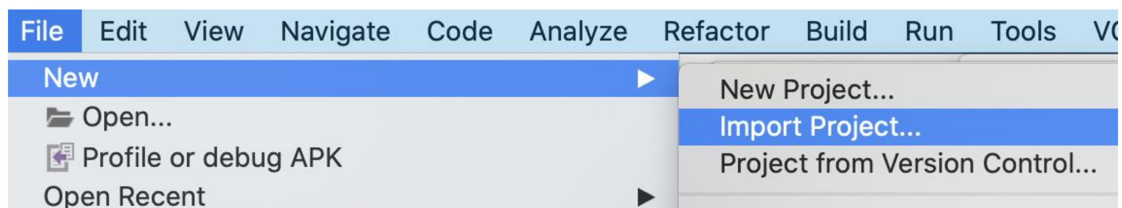
3. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
4. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
5. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

Otwórz projekt w Android Studio

1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz istniejący projekt Android Studio.**



Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Nowy** > **Importuj projekt** .



3. W oknie dialogowym **Importuj projekt** przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
4. Kliknij dwukrotnie ten folder projektu.

5. Poczekaj, aż Android Studio otworzy projekt.
6. Kliknij przycisk **Uruchom** , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.
7. Przeglądaj pliki projektu w oknie narzędzia **Projekt** , aby zobaczyć, jak skonfigurowana jest aplikacja.

Przejdź przez kod startowy

1. Otwórz pobrany projekt w Android Studio. Nazwa folderu projektu to `android-basics-kotlin-cupcake-app-starter`. Następnie uruchom aplikację.
2. Przeglądaj pliki, aby zrozumieć kod startowy. W przypadku plików układu możesz użyć opcji **Podziel** w prawym górnym rogu, aby jednocześnie wyświetlić podgląd układu i XML.
3. Po skompilowaniu i uruchomieniu aplikacji zauważysz, że aplikacja jest niekompletna. Przyciski nie robią wiele (poza wyświetlaniem `Toast`wiadomości) i nie można nawigować do innych fragmentów.

   Oto przewodnik po ważnych plikach w projekcie.

Główna aktywność:

Ma `MainActivity`podobny kod do domyślnego wygenerowanego kodu, który ustawia widok zawartości działania jako `activity_main.xml`. Ten kod używa sparametryzowanego konstruktora `AppCompatActivity(@LayoutRes int contentLayoutId)`, który przyjmuje układ, który zostanie zawyżony jako część `super.onCreate(savedInstanceState)`.

Kod w `MainActivity`klasie

```
class MainActivity : AppCompatActivity(R.layout.activity_main)
```

jest taki sam jak poniższy kod przy użyciu domyślnego `AppCompatActivity`konstruktora:

```
class MainActivity : AppCompatActivity() {

  override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
  }
}
```

Układy (folder res/layout):

Folder `layout`zasobów zawiera pliki aktywności i fragmenty układu. Są to proste pliki układu, a XML jest znany z poprzednich ćwiczeń z programowania.

- `fragment_start.xml`to pierwszy ekran wyświetlany w aplikacji. Posiada obrazek babeczki i trzy przyciski do wyboru liczby babeczek do zamówienia: jedna babeczka, sześć babeczek i dwanaście babeczek.

- fragment_flavor.xmlpokazuje listę smaków babeczek jako opcje przycisków radiowych z przyciskiem **Dalej** .
- fragment_pickup.xmludostępnia opcję wyboru dnia odbioru i przycisk **Dalej** , aby przejść do ekranu podsumowania.
- fragment_summary.xmlwyświetla podsumowanie szczegółów zamówienia, takich jak ilość, smak oraz przycisk do wysłania zamówienia do innej aplikacji.

Klasy fragmentów:

- StartFragment.ktto pierwszy ekran wyświetlany w aplikacji. Ta klasa zawiera kod powiązania widoku i procedurę obsługi kliknięcia dla trzech przycisków.
- FlavorFragment.kt, PickupFragment.kt, i SummaryFragment.ktklasy zawierają głównie standardowy kod i procedurę obsługi kliknięcia przycisku **Dalej** lub **Wyślij zamówienie do innej aplikacji** , które wyświetlają komunikat wyskakujący.

Zasoby (folder res):

- drawablefolder zawiera zasób babeczki dla pierwszego ekranu, a także pliki ikon programu uruchamiającego.
- navigation/nav_graph.xmlzawiera cztery miejsca docelowe fragmentów (startFragment, flavorFragment, pickupFragmenti summaryFragment) bez *akcji* , które zdefiniujesz później w ćwiczeniu z programowania.
- valuesfolder zawiera kolory, wymiary, ciągi, style i motywy używane do dostosowywania motywu aplikacji. Powinieneś znać te typy zasobów z poprzednich ćwiczeń z programowania.

3. Uzupełnij wykres nawigacji

W tym zadaniu połączysz ze sobą ekrany aplikacji **Cupcake** i dokończysz wdrażanie prawidłowej nawigacji w aplikacji.

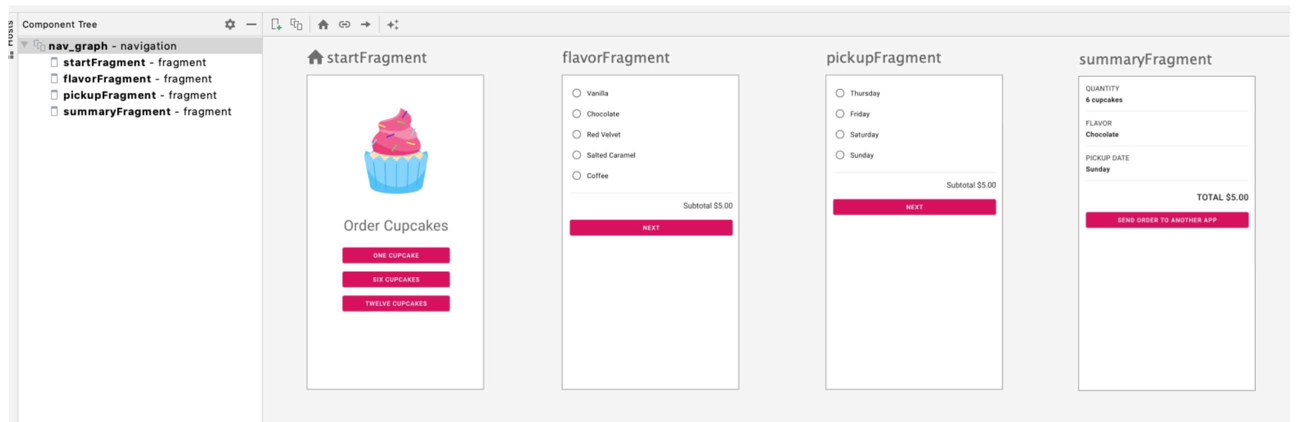
Czy pamiętasz, czego potrzebujemy, aby korzystać z komponentu Nawigacja? Postępuj zgodnie z tym przewodnikiem , aby dowiedzieć się, jak skonfigurować projekt i aplikację, aby:

- Dołącz bibliotekę Jetpack Navigation
- Dodaj a NavHostdo aktywności
- Utwórz wykres nawigacyjny
- Dodaj miejsca docelowe fragmentów do wykresu nawigacyjnego

Połącz miejsca docelowe na wykresie nawigacji

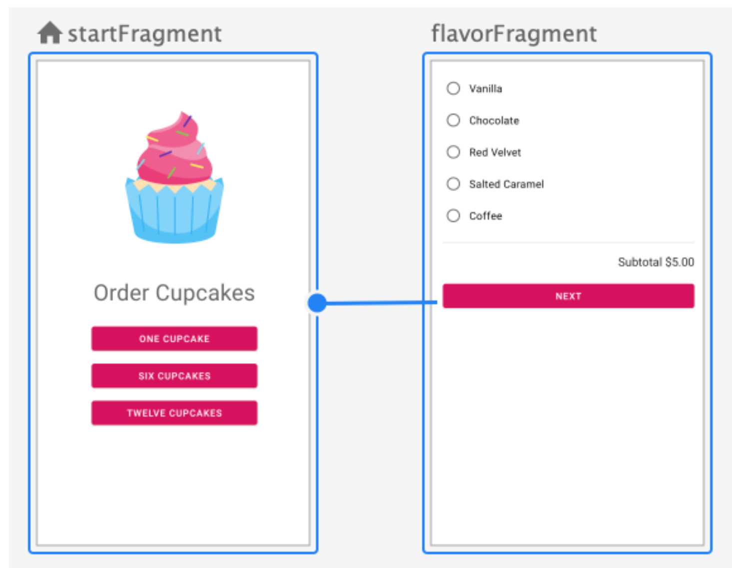
1. W Android Studio w oknie **Project** otwórz **res > navigation >** plik nav_graph.xml. Przejdź do karty **Projekt** , jeśli nie została jeszcze wybrana.

2. Spowoduje to otwarcie **Edytora nawigacji** w celu wizualizacji wykresu nawigacji w Twojej aplikacji. Powinieneś zobaczyć cztery fragmenty, które już istnieją w aplikacji.

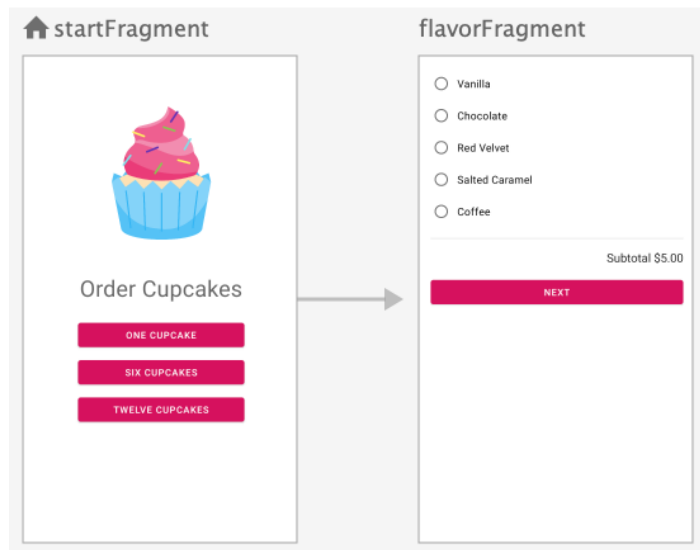


Uwaga: Jeśli fragmenty docelowe są inaczej ułożone w Android Studio, kliknij i przeciągnij miejsca docelowe, aby zmienić ich kolejność, podobnie jak na powyższym zrzucie ekranu. Ułatwia to późniejsze konfigurowanie akcji nawigacji w laboratorium programowania.

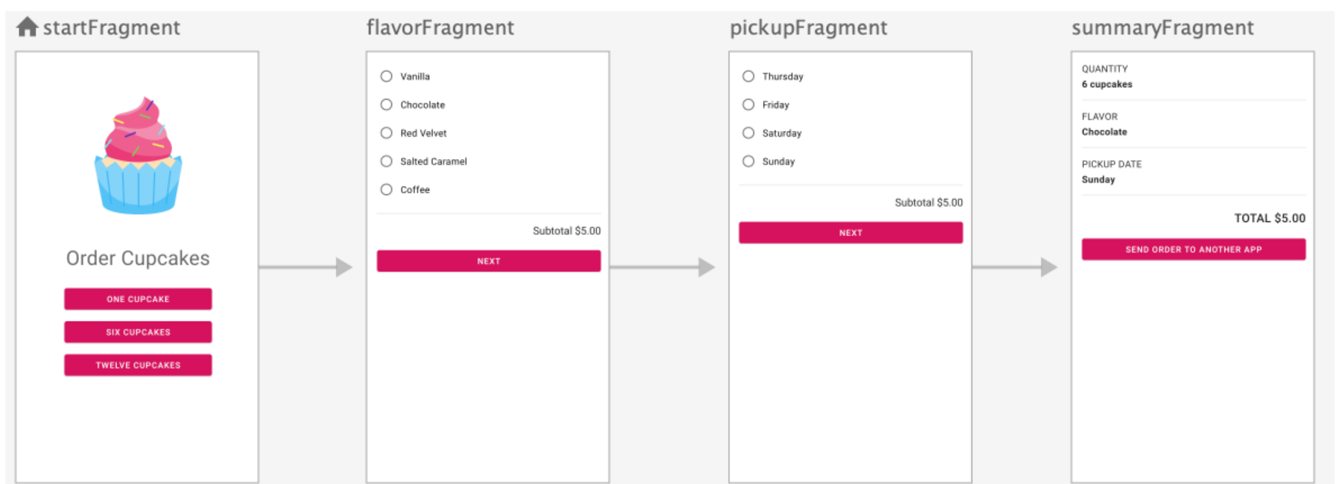
3. Połącz miejsca docelowe fragmentów na wykresie nawigacji. Utwórz akcję z startFragmentdo flavorFragment, połączenie z flavorFragmentdo pickupFragmenti połączenie z pickupFragmentdo summaryFragment. Wykonaj kilka następnych kroków, jeśli potrzebujesz bardziej szczegółowych instrukcji.
4. Najedź kursorem na **startFragment** , aż zobaczysz szarą ramkę wokół fragmentu, a nad środkiem prawej krawędzi fragmentu pojawi się szare kółko. Kliknij kółko i przeciągnij do **smakuFragment** , a następnie zwolnij mysz.



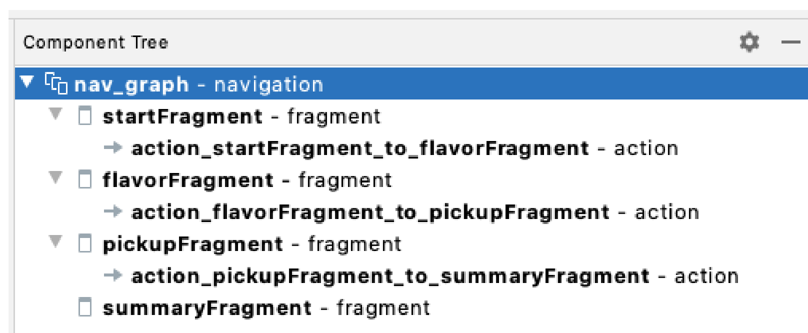
5. Strzałka między tymi dwoma fragmentami wskazuje pomyślne połączenie, co oznacza, że będziesz mógł przejść od **startFragment** do **smakuFragment** . Nazywa się to akcją nawigacji, której nauczyłeś się podczas poprzedniego ćwiczenia z programowania.



6. Podobnie dodaj akcje nawigacyjne od **smakuFragment** do **pickupFragment** i od **pickupFragment** do **summaryFragment** . Po zakończeniu tworzenia akcji nawigacji ukończony wykres nawigacji powinien wyglądać następująco.

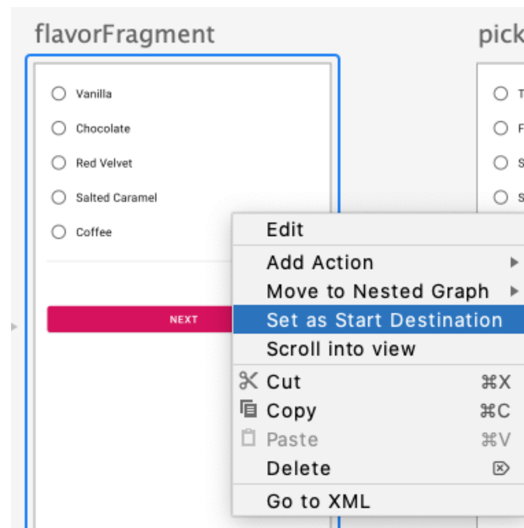


7. Trzy nowe akcje, które utworzyłeś, powinny być również odzwierciedlone w panelu **Drzewo komponentów** .



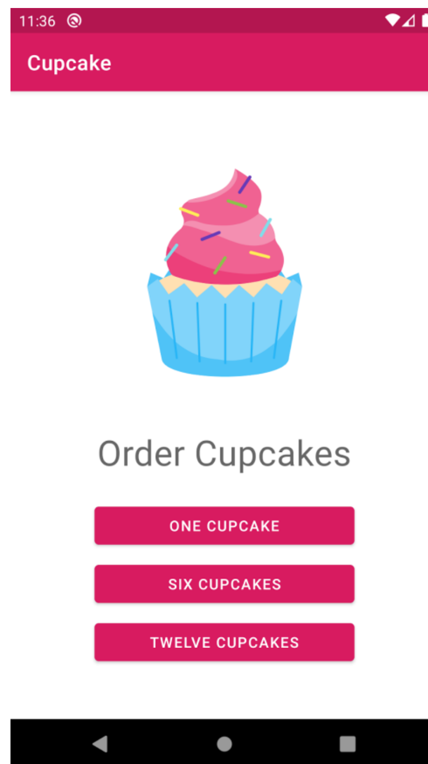
8. Definiując wykres nawigacyjny, chcesz również określić miejsce docelowe. Obecnie widzisz, że startFragment ma obok siebie małą ikonę domu.

Oznacza to, że **startFragment** będzie pierwszym fragmentem, który zostanie pokazany w NavHost. Pozostaw to jako pożądane zachowanie naszej aplikacji. Na przyszłość zawsze możesz zmienić miejsce docelowe, klikając prawym przyciskiem myszy fragment i wybierając opcję menu **Ustaw jako miejsce docelowe początkowe** .



Przejdź od fragmentu początkowego do fragmentu smakowego

Następnie dodasz kod, aby przejść od **startFragment** do **smakuFragment** , dotykając przycisków w pierwszym fragmencie, zamiast wyświetlać Toastkomunikat. Poniżej znajduje się odniesienie do układu fragmentów początkowych. Liczbę babeczek przekażesz do fragmentu smaku w późniejszym zadaniu.



1. W oknie **projektu** otwórz **aplikację > java > com.example.cupcake >** plik StartFragment Kotlin.

2. W tej onViewCreated()metodzie zauważ, że detektory kliknięć są ustawione na trzech przyciskach. Po naciśnięciu każdego przycisku orderCupcake()metoda jest wywoływana z liczbą babeczek (1, 6 lub 12 babeczek) jako parametrem.

   Kod referencyjny:

```
orderOneCupcake.setOnClickListener { orderCupcake(1) }
orderSixCupcakes.setOnClickListener { orderCupcake(6) }
orderTwelveCupcakes.setOnClickListener { orderCupcake(12) }
```

3. W orderCupcake()metodzie zastąp kod wyświetlający wiadomość wyskakującą kodem, aby przejść do fragmentu smaku. Pobierz metodę NavControllerusing findNavController()i wywołaj navigate()ją, przekazując identyfikator akcji, R.id.action_startFragment_to_flavorFragment. Upewnij się, że ten identyfikator akcji pasuje do akcji zadeklarowanej w twoimnav_graph.xml.

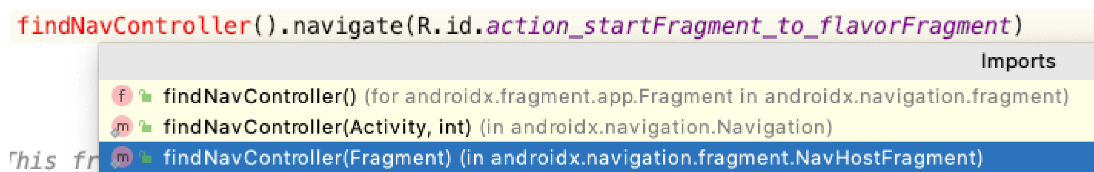
   Zastępować

```
fun orderCupcake(quantity: Int) {
    Toast.makeText(activity, "Ordered $quantity cupcake(s)", Toast.LENGTH_SHORT).show()
}
```

   z

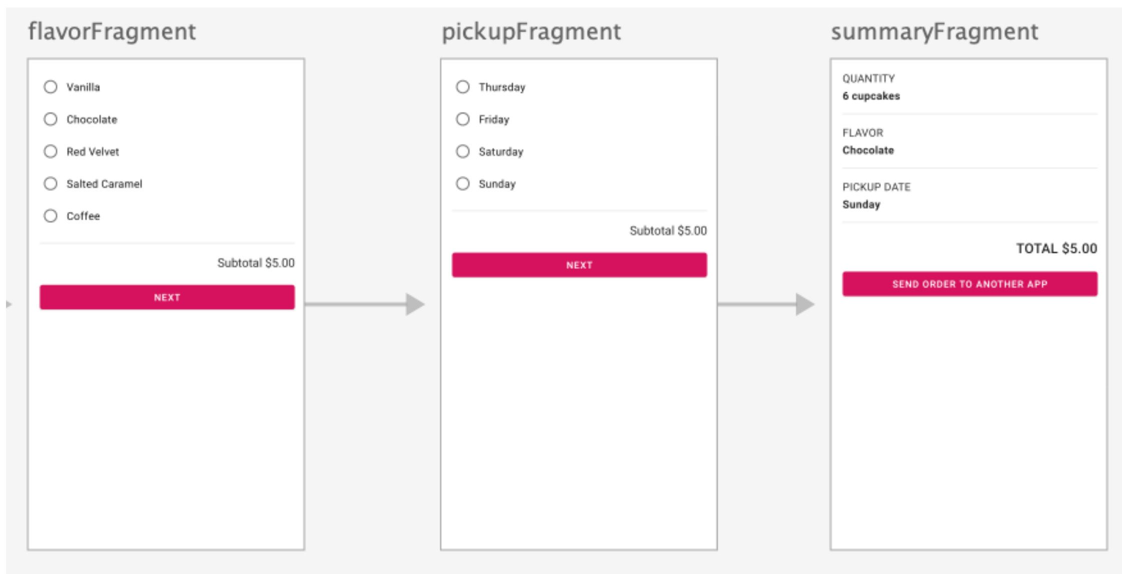
```
fun orderCupcake(quantity: Int) {
   findNavController().navigate(R.id.action_startFragment_to_flavorFragment)
}
```

4. Dodaj import import androidx.navigation.fragment.findNavControllerlub wybierz jedną z opcji dostarczanych przez Android Studio.



Dodaj Nawigację do smaku i fragmentów odbioru

Podobnie jak w poprzednim zadaniu, w tym zadaniu dodasz nawigację do pozostałych fragmentów: smaku i fragmentów odbioru.



1. Otwórz **aplikację > java > com.example.cupcake > FlavorFragment.kt** . Zwróć uwagę, że metoda wywoływana w odbiorniku kliknięcia przycisku **Dalej**goToNextScreen() jest metodą.
2. W FlavorFragment.kt, wewnątrz goToNextScreen()metody, zastąp kod wyświetlający toast, aby przejść do fragmentu odbioru. Użyj identyfikatora akcji R.id.action_flavorFragment_to_pickupFragmenti upewnij się, że jest on zgodny z akcją zadeklarowaną w twoimnav_graph.xml.

```
fun goToNextScreen() {
   findNavController().navigate(R.id.action_flavorFragment_to_pickupFragment)
}
```

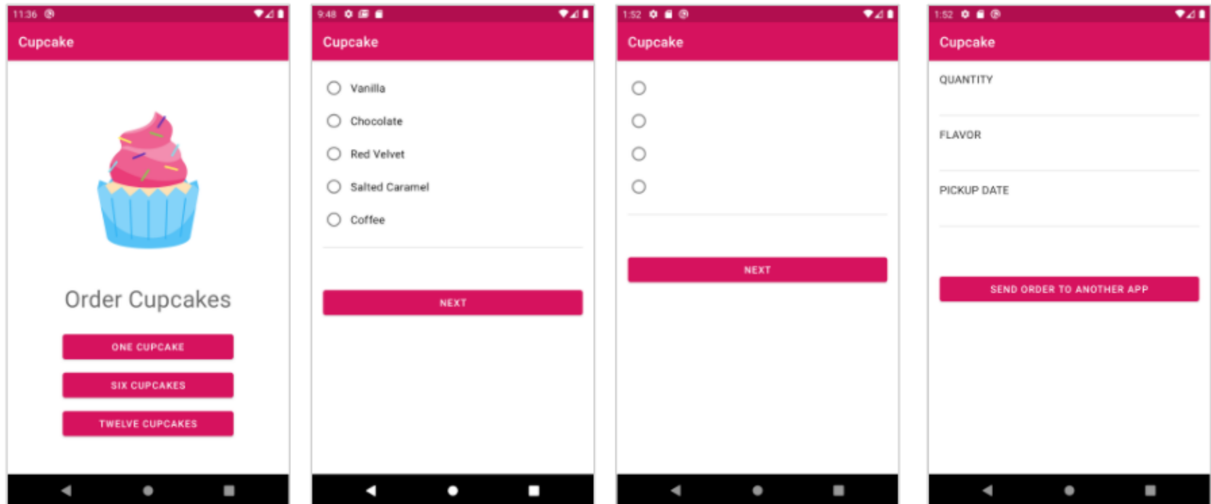
   Pamiętaj, aby import androidx.navigation.fragment.findNavController.

3. Podobnie w PickupFragment.kt, wewnątrz goToNextScreen()metody, zastąp istniejący kod, aby przejść do fragmentu podsumowania.

```
fun goToNextScreen() {
   findNavController().navigate(R.id.action_pickupFragment_to_summaryFragment)
}
```

   Importuj androidx.navigation.fragment.findNavController.

4. Uruchom aplikację. Upewnij się, że przyciski działają podczas nawigacji z ekranu na ekran. Informacje wyświetlane na każdym fragmencie mogą być niekompletne, ale nie martw się, w kolejnych krokach wypełnisz te fragmenty poprawnymi danymi.

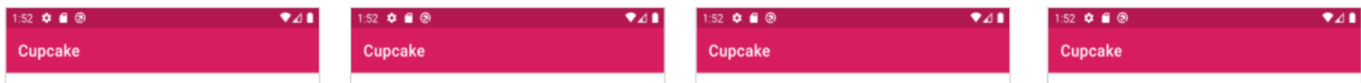


Zaktualizuj tytuł na pasku aplikacji

Podczas poruszania się po aplikacji zwróć uwagę na tytuł na pasku aplikacji. Jest zawsze wyświetlany jako **Cupcake.**

Lepszym doświadczeniem użytkownika byłoby podanie bardziej odpowiedniego tytułu w oparciu o funkcjonalność bieżącego fragmentu.

Zmień tytuł na pasku aplikacji (znanym również jako pasek akcji) dla każdego fragmentu za pomocą przycisku NavControlleri wyświetl przycisk **W górę** (←).



1. W MainActivity.ktprogramie zastąp onCreate()metodę konfiguracji kontrolera nawigacyjnego. Pobierz instancję NavControllerz NavHostFragment.
2. Zadzwoń do setupActionBarWithNavController(navController)przekazywania w instancji NavController. Spowoduje to następujące czynności: Pokaż tytuł na pasku aplikacji oparty na etykiecie miejsca docelowego i wyświetl przycisk **W górę** , gdy nie znajdujesz się w miejscu docelowym najwyższego poziomu.

```
class MainActivity : AppCompatActivity(R.layout.activity_main) {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val navHostFragment = supportFragmentManager
                .findFragmentById(R.id.nav_host_fragment) as NavHostFragment
        val navController = navHostFragment.navController

        setupActionBarWithNavController(navController)
    }
}
```

3. Dodaj niezbędne importy po wyświetleniu monitu przez Android Studio.

```
import android.os.Bundle
import androidx.navigation.fragment.NavHostFragment
import androidx.navigation.ui.setupActionBarWithNavController
```

4. Ustaw tytuły paska aplikacji dla każdego fragmentu. Otwórz `navigation/nav_graph.xml`i przejdź do zakładki **Kod .**
5. W `nav_graph.xml`programie zmodyfikuj `android:label`atrybut dla każdego miejsca docelowego fragmentu. Użyj następujących zasobów ciągów, które zostały już zadeklarowane w aplikacji startowej.

   Dla fragmentu początkowego użyj `@string/app_name`z value `Cupcake`.

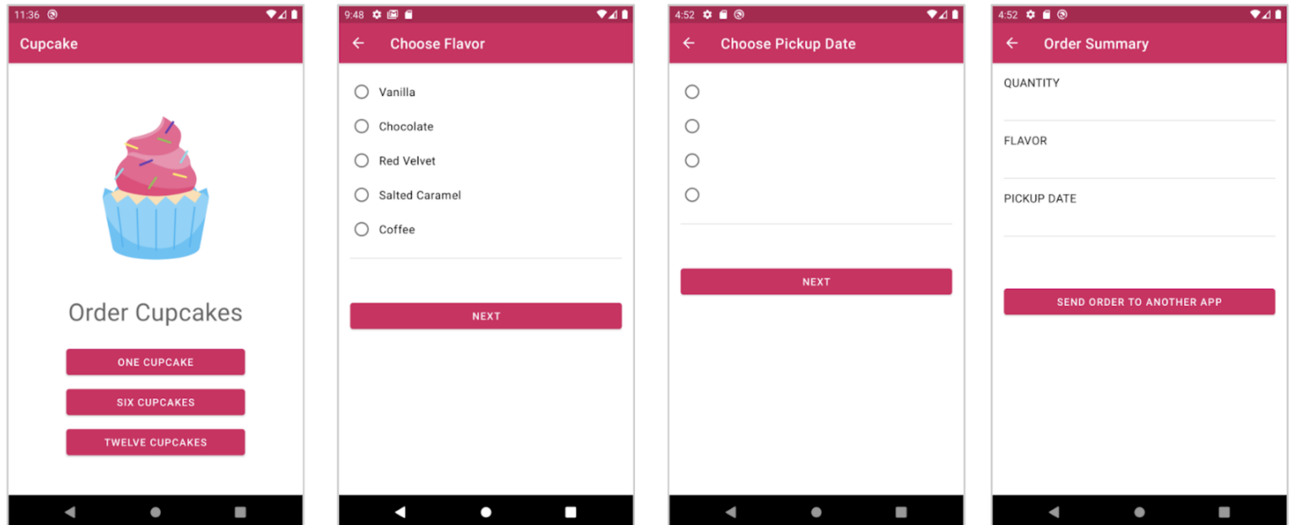
   Dla fragmentu smaku użyj `@string/choose_flavor`z wartością `Choose Flavor`.

   W przypadku fragmentu odbioru użyj `@string/choose_pickup_date`z wartością `Choose Pickup Date`.

   W przypadku fragmentu podsumowania użyj `@string/order_summary`z wartością `Order Summary`.

```
<navigation ...>
   <fragment
      android:id="@+id/startFragment"
      ...
      android:label="@string/app_name" ... >
      <action ... />
   </fragment>
   <fragment
      android:id="@+id/flavorFragment"
      ...
      android:label="@string/choose_flavor" ... >
      <action ... />
   </fragment>
   <fragment
      android:id="@+id/pickupFragment"
      ...
      android:label="@string/choose_pickup_date" ... >
      <action ... />
   </fragment>
   <fragment
      android:id="@+id/summaryFragment"
      ...
      android:label="@string/order_summary" ... />
</navigation>
```

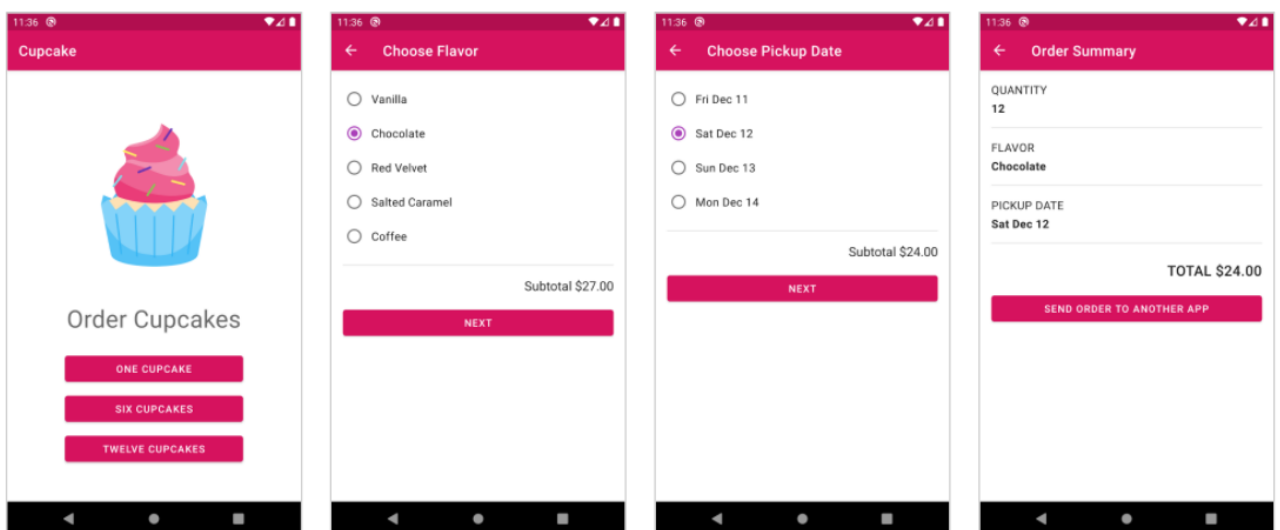
6. Uruchom aplikację. Zwróć uwagę, że tytuł na pasku aplikacji zmienia się, gdy przechodzisz do każdego miejsca docelowego fragmentu. Zauważ również, że przycisk **W górę** (strzałka ←) jest teraz wyświetlany na pasku aplikacji. Jeśli go dotkniesz, nic nie zrobi. Zaimplementujesz zachowanie przycisku **W górę** w następnym ćwiczeniu z programowania.



4. Utwórz udostępniony ViewModel

Przejdźmy do wypełniania poprawnych danych w każdym z fragmentów. Będziesz używać udostępnionego `ViewModel`do zapisywania danych aplikacji w jednym `ViewModel`. Wiele fragmentów w aplikacji będzie uzyskiwać dostęp do udostępnionego `ViewModel`przy użyciu ich zakresu aktywności.

Powszechnym przypadkiem użycia jest udostępnianie danych między fragmentami w większości aplikacji produkcyjnych. Na przykład w ostatecznej wersji (tego ćwiczenia z kodowania) aplikacji **Cupcake** (zwróć uwagę na zrzuty ekranu poniżej) użytkownik wybiera ilość babeczek na pierwszym ekranie, a na drugim ekranie cena jest obliczana i wyświetlana na podstawie ilości babeczki. Podobnie inne dane aplikacji, takie jak smak i data odbioru, są również używane na ekranie podsumowania.



Patrząc na funkcje aplikacji, możesz stwierdzić, że przydatne byłoby przechowywanie tych informacji o zamówieniu w jednym `ViewModel`, które można udostępniać we fragmentach tego

działania. Przypomnij sobie, że ViewModeljest to część składników architektury systemu Android . Dane aplikacji zapisane w aplikacji ViewModelsą zachowywane podczas zmian konfiguracji. Aby dodać a ViewModeldo swojej aplikacji, tworzysz nową klasę, która dziedziczy z ViewModelklasy.

Utwórz model widoku zamówienia

W tym zadaniu utworzysz udostępnioną ViewModelaplikację **Cupcake** o nazwie OrderViewModel. Dodasz również dane aplikacji jako właściwości w ViewModelmetodach i w celu aktualizacji i modyfikacji danych. Oto właściwości klasy:

- Ilość zamówienia (Integer)
- Smak babeczki (String)
- Data odbioru (String)
- Cena (Double)

Postępuj zgodnie z ViewModelnajlepszymi praktykami

W przypadku ViewModelzalecaną praktyką jest *nieuwidacznianie* danych widoku modelu jako publiczmiennych. W przeciwnym razie dane aplikacji mogą zostać zmodyfikowane w nieoczekiwany sposób przez klasy zewnętrzne i utworzyć przypadki brzegowe, których aplikacja nie spodziewała się obsłużyć. Zamiast tego utwórz te zmienne właściwości private, zaimplementuj właściwość kopii zapasowej i w publicrazie potrzeby uwidocznij niezmienną wersję każdej właściwości. Konwencja polega na poprzedzeniu nazwy private właściwości mutowalnych znakiem podkreślenia (_).

Oto metody aktualizacji powyższych właściwości, w zależności od wyboru użytkownika:

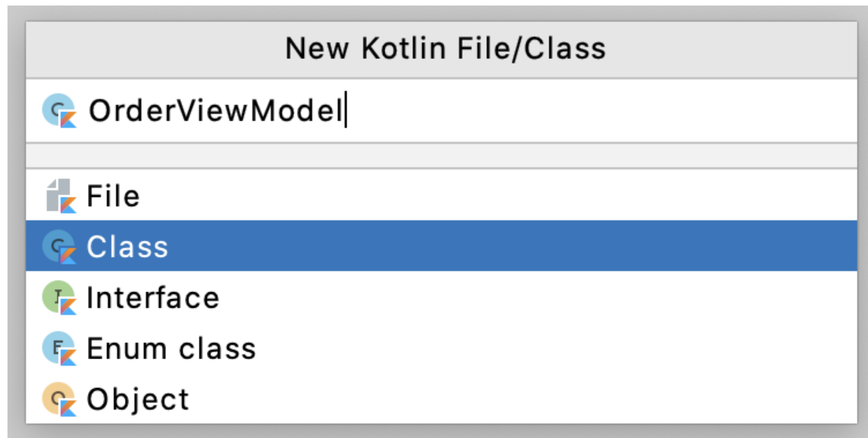
- setQuantity(numberCupcakes: Int)
- setFlavor(desiredFlavor: String)
- setDate(pickupDate: String)

Nie potrzebujesz metody ustalającej cenę, ponieważ obliczysz ją w ramach OrderViewModelinnych właściwości. Poniższe kroki przeprowadzą Cię przez proces implementacji udostępnionego ViewModel.

Utworzysz nowy pakiet w swoim projekcie o nazwie modeli dodasz OrderViewModelklasę. Spowoduje to oddzielenie kodu modelu widoku od reszty kodu interfejsu użytkownika (fragmentów i działań). Najlepszą praktyką kodowania jest podzielenie kodu na pakiety w zależności od funkcjonalności.

1. W oknie **projektu** Android Studio kliknij prawym przyciskiem myszy **com.example.cupcake > New > Package** .
2. Otworzy się okno dialogowe **Nowy pakiet** , podaj nazwę pakietu jako com.example.cupcake.model.

3. Utwórz OrderViewModelklasę Kotlin pod modelpakietem. W oknie **Project** kliknij prawym przyciskiem myszy modelpakiet i wybierz **New > Kotlin File/Class** . W nowym oknie podaj nazwę pliku OrderViewModel.



4. W OrderViewModel.ktprogramie zmień sygnaturę klasy na rozszerzenie ViewModel.

```
import androidx.lifecycle.ViewModel

class OrderViewModel : ViewModel() {

}
```

5. Wewnątrz OrderViewModelklasy dodaj właściwości, które zostały omówione powyżej jako private val.
6. Zmień typy właściwości LiveDatai dodaj pola zapasowe do właściwości, aby można było zaobserwować te właściwości i zaktualizować interfejs użytkownika, gdy zmienią się dane źródłowe w modelu widoku.

```
private val _quantity = MutableLiveData<Int>(0)
val quantity: LiveData<Int> = _quantity

private val _flavor = MutableLiveData<String>("")
val flavor: LiveData<String> = _flavor

private val _date = MutableLiveData<String>("")
val date: LiveData<String> = _date

private val _price = MutableLiveData<Double>(0.0)
val price: LiveData<Double> = _price
```

Będziesz musiał zaimportować te klasy:

```
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
```

7. Na OrderViewModelzajęciach dodaj metody, które zostały omówione powyżej. Wewnątrz metod przypisz argument przekazany do właściwości mutable.
8. Ponieważ te metody ustawiające muszą być wywoływane spoza modelu widoku, pozostaw je jako publicmetody (co oznacza brak privatelub inny modyfikator widoczności wymagany przed funsłowem kluczowym). Domyślny modyfikator widoczności w Kotlin to public.

```
fun setQuantity(numberCupcakes: Int) {
    _quantity.value = numberCupcakes
}

fun setFlavor(desiredFlavor: String) {
    _flavor.value = desiredFlavor
}

fun setDate(pickupDate: String) {
    _date.value = pickupDate
}
```

9. Skompiluj i uruchom swoją aplikację, aby upewnić się, że nie ma błędów kompilacji. W interfejsie użytkownika nie powinno być jeszcze żadnych widocznych zmian.

   Dobra robota! Teraz masz początek swojego modelu widoku. Będziesz stopniowo dodawać więcej do tej klasy, gdy będziesz tworzyć więcej funkcji w swojej aplikacji i zdasz sobie sprawę, że potrzebujesz więcej właściwości i metod w swojej klasie.

   Jeśli widzisz nazwy klas, nazwy właściwości lub nazwy metod w szarej czcionce w Android Studio, jest to oczekiwane. Oznacza to, że klasa, właściwości lub metody lub nie są w tej chwili używane, ale będą! To będzie następne.

5. Użyj ViewModel, aby zaktualizować interfejs użytkownika

W tym zadaniu użyjesz utworzonego modelu widoku współdzielonego, aby zaktualizować interfejs użytkownika aplikacji. Główną różnicą w implementacji modelu widoku współdzielonego jest sposób, w jaki uzyskujemy do niego dostęp z kontrolerów interfejsu użytkownika. Użyjesz instancji aktywności zamiast instancji fragmentu i zobaczysz, jak to zrobić w kolejnych sekcjach.

Oznacza to, że model widoku można udostępniać we fragmentach. Każdy fragment może uzyskać dostęp do modelu widoku, aby sprawdzić niektóre szczegóły zamówienia lub zaktualizować niektóre dane w modelu widoku.

Zaktualizuj StartFragment, aby użyć modelu widoku

Aby użyć udostępnionego modelu widoku `StartFragment`, zainicjujesz `OrderViewModel`using `activityViewModels()`zamiast `viewModels()`delegata.

- `viewModels()`daje `ViewModel`instancję z zakresem do bieżącego fragmentu. To będzie inne dla różnych fragmentów.
- `activityViewModels()`udostępnia `ViewModel`instancję objętą zakresem bieżącego działania. Dlatego instancja pozostanie taka sama w wielu fragmentach tej samej aktywności.

Użyj delegata właściwości Kotlin

W Kotlinie każda `var`właściwość mutable ( ) ma domyślne funkcje pobierające i ustawiające automatycznie generowane dla niej. Funkcje ustawiające i pobierające są wywoływane podczas przypisywania wartości lub odczytywania wartości właściwości. (W przypadku właściwości tylko

do odczytu (val) domyślnie generowana jest tylko funkcja pobierająca. Ta funkcja pobierająca jest wywoływana podczas odczytywania wartości właściwości tylko do odczytu.)

Delegacja własności w Kotlinie pomaga przekazać odpowiedzialność pobierającego setera innej klasie.

Ta klasa (zwana *klasą delegata*) udostępnia funkcje pobierające i ustawiające właściwości oraz obsługuje jej zmiany.

Właściwość delegata jest definiowana za pomocą byklauzuli i instancji klasy delegata:

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

1. W StartFragmentklasie pobierz odwołanie do modelu widoku współdzielonego jako zmienną klasy. Użyj by activityViewModels()delegata właściwości Kotlin z fragment-ktxbiblioteki.

   ```
   private val sharedViewModel: OrderViewModel by activityViewModels()
   ```

   Możesz potrzebować tych nowych importów:

   ```
   import androidx.fragment.app.activityViewModels
   import com.example.cupcake.model.OrderViewModel
   ```

2. Powtórz powyższy krok dla klas FlavorFragment, PickupFragment, SummaryFragmentbędziesz używać tego sharedViewModelwystąpienia w dalszych sekcjach ćwiczenia z programowania.
3. Wracając do StartFragmentzajęć, możesz teraz korzystać z modelu widoku. Na początku orderCupcake()metody wywołaj setQuantity()metodę w modelu widoku współdzielonego, aby zaktualizować ilość, przed przejściem do fragmentu smaku.

   ```
   fun orderCupcake(quantity: Int) {
      sharedViewModel.setQuantity(quantity)
      findNavController().navigate(R.id.action_startFragment_to_flavorFragment)
   }
   ```

4. W ramach OrderViewModelklasy dodaj następującą metodę, aby sprawdzić, czy smak zamówienia został ustawiony, czy nie. Użyjesz tej metody na StartFragmentzajęciach w późniejszym kroku.

   ```
   fun hasNoFlavorSet(): Boolean {
      return _flavor.value.isNullOrEmpty()
   }
   ```

5. W StartFragmentklasie, orderCupcake()metoda wewnętrzna, po ustawieniu ilości, ustaw domyślny smak jako Waniliowy, jeśli nie jest ustawiony żaden smak, przed przejściem do fragmentu smaku. Twoja pełna metoda będzie wyglądać tak:

   ```
   fun orderCupcake(quantity: Int) {
      sharedViewModel.setQuantity(quantity)
      if (sharedViewModel.hasNoFlavorSet()) {
         sharedViewModel.setFlavor(getString(R.string.vanilla))
      }
   ```

```
    findNavController().navigate(R.id.action_startFragment_to_flavorFragment)
}
```

6. Skompiluj aplikację, aby upewnić się, że nie ma błędów kompilacji. Jednak nie powinno być żadnych widocznych zmian w twoim interfejsie użytkownika.

6. Użyj ViewModel z wiązaniem danych

Następnie użyjesz powiązania danych, aby powiązać dane modelu widoku z interfejsem użytkownika. Zaktualizujesz również model widoku udostępnionego na podstawie wyborów dokonanych przez użytkownika w interfejsie użytkownika.

Odświeżenie na temat wiązania danych

Przypomnij sobie, że [biblioteka powiązań danych](#) jest częścią pakietu [Android Jetpack](#) . Powiązanie danych wiąże składniki interfejsu użytkownika w układach ze źródłami danych w aplikacji przy użyciu formatu deklaratywnego. Mówiąc prościej, wiązanie danych to wiązanie danych (z kodu) z widokami + wiązanie widoków (wiązanie widoków z kodem). Dzięki skonfigurowaniu tych powiązań i automatycznym aktualizacjom pomaga to zmniejszyć ryzyko wystąpienia błędów, jeśli zapomnisz ręcznie zaktualizować interfejs użytkownika z poziomu kodu.

Zaktualizuj smak z wyborem użytkownika

1. W programie `layout/fragment_flavor.xml`dodaj `<data>`tag wewnątrz tagu głównego `<layout>`. Dodaj zmienną układu o nazwie `viewModel`typu `com.example.cupcake.model.OrderViewModel`. Upewnij się, że nazwa pakietu w atrybucie type jest zgodna z nazwą pakietu klasy modelu widoku udostępnionego `OrderViewModel`w Twojej aplikacji.

```
<layout ...>

    <data>
        <variable
            name="viewModel"
            type="com.example.cupcake.model.OrderViewModel" />
    </data>

    <ScrollView ...>

    ...
```

2. Podobnie powtórz powyższy krok dla `fragment_pickup.xml`i , `fragment_summary.xml`aby dodać `viewModel`zmienną układu. Użyjesz tej zmiennej w późniejszych rozdziałach. Nie musisz dodawać tego kodu w `fragment_start.xml`programie , ponieważ ten układ nie używa modelu widoku udostępnionego.
3. W `FlavorFragment`klasie wewnątrz `onViewCreated()`powiąż instancję modelu widoku z instancją modelu widoku współdzielonego w układzie. Dodaj następujący kod wewnątrz `binding?.apply`bloku.

```
binding?.apply {
    viewModel = sharedViewModel
    ...
```

```
}
```

Zastosuj funkcję zakresu

Może to być pierwszy raz, kiedy widzisz tę `apply`funkcję w Kotlinie. *`apply`*to [funkcja zasięgu](#) w standardowej bibliotece Kotlin. Wykonuje blok kodu w kontekście obiektu. Tworzy tymczasowy zakres iw tym zakresie możesz uzyskać dostęp do obiektu bez jego nazwy. Typowym przypadkiem użycia for `apply`jest skonfigurowanie obiektu. Takie wywołania można odczytać jako *„zastosuj następujące przypisania do obiektu”.*

Przykład:

```
clark.apply {
   firstName = "Clark"
   lastName = "James"
   age = 18
}

// The equivalent code without apply scope function would look like the following.

clark.firstName = "Clark"
clark.lastName = "James"
clark.age = 18
```

4. Powtórz ten sam krok dla `onViewCreated()`metody wewnątrz klas `PickupFragment`i .`SummaryFragment`

```
binding?.apply {
   viewModel = sharedViewModel
   ...
}
```

5. W `fragment_flavor.xml`programie użyj nowej zmiennej układu, `viewModel`aby ustawić `checked`atrybut przycisków opcji na podstawie `flavor`wartości w modelu widoku. Jeśli smak reprezentowany przez przycisk opcji jest taki sam, jak smak zapisany w modelu widoku, wyświetl przycisk opcji jako zaznaczony (`checked`= true). Wyrażenie wiążące dla sprawdzanego stanu **Vanilla**`RadioButton` wyglądałoby następująco:

`@{viewModel.flavor.equals(@string/vanilla)}`

Zasadniczo porównujesz `viewModel.flavor`właściwość z odpowiednim zasobem ciągu za pomocą [`equals`](#)funkcji, aby określić, czy zaznaczony stan powinien być prawdziwy, czy fałszywy.

Uwaga: pamiętaj, że wyrażenia wiążące zaczynają się od `@`symbolu i są ujęte w nawiasy klamrowe `{}`.

```
<RadioGroup
  ...>

  <RadioButton
     android:id="@+id/vanilla"
     ...
```

```
    android:checked="@{viewModel.flavor.equals(@string/vanilla)}"
    .../>

  <RadioButton
    android:id="@+id/chocolate"
    ...
    android:checked="@{viewModel.flavor.equals(@string/chocolate)}"
    .../>

  <RadioButton
    android:id="@+id/red_velvet"
    ...
    android:checked="@{viewModel.flavor.equals(@string/red_velvet)}"
    .../>

  <RadioButton
    android:id="@+id/salted_caramel"
    ...
    android:checked="@{viewModel.flavor.equals(@string/salted_caramel)}"
    .../>

  <RadioButton
    android:id="@+id/coffee"
    ...
    android:checked="@{viewModel.flavor.equals(@string/coffee)}"
    .../>
</RadioGroup>
```

Wiązania słuchacza

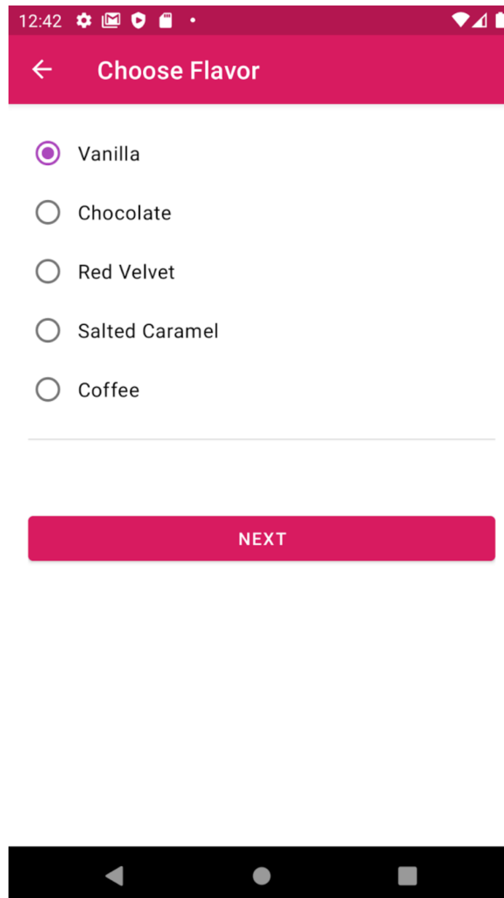
Powiązania odbiornika to wyrażenia lambda, które są uruchamiane, gdy wystąpi zdarzenie, takie jak `onClick`zdarzenie. Są one podobne do odwołań do metod, na przykład `textview.setOnClickListener(clickListener)`powiązania odbiornika umożliwiają uruchamianie dowolnych wyrażeń powiązania danych.

1. W `fragment_flavor.xml`programie dodaj detektory zdarzeń do przycisków radiowych za pomocą powiązań detektorów. Użyj wyrażenia lambda bez parametrów i wywołaj `viewModel`. `setFlavor()`metody, przekazując odpowiedni zasób ciągu smakowego.

```
<RadioGroup
  ...>

  <RadioButton
    android:id="@+id/vanilla"
    ...
    android:onClick="@{() -> viewModel.setFlavor(@string/vanilla)}"
    .../>
```

```
    <RadioButton
        android:id="@+id/chocolate"
        ...
        android:onClick="@{() -> viewModel.setFlavor(@string/chocolate)}"
        .../>

    <RadioButton
        android:id="@+id/red_velvet"
        ...
        android:onClick="@{() -> viewModel.setFlavor(@string/red_velvet)}"
        .../>

    <RadioButton
        android:id="@+id/salted_caramel"
        ...
        android:onClick="@{() -> viewModel.setFlavor(@string/salted_caramel)}"
        .../>

    <RadioButton
        android:id="@+id/coffee"
        ...
        android:onClick="@{() -> viewModel.setFlavor(@string/coffee)}"
        .../>
</RadioGroup>
```

7. Uruchom aplikację i zauważ, jak domyślnie wybrana jest opcja **Vanilla we fragmencie smaku.**



Świetny! Teraz możesz przejść do kolejnych fragmentów.

7. Zaktualizuj fragment odbioru i podsumowania, aby użyć modelu widoku

Poruszaj się po aplikacji i zauważ, że we fragmencie odbioru etykiety opcji przycisków radiowych są puste. W tym zadaniu obliczysz 4 dostępne terminy odbioru i wyświetlisz je we fragmencie odbioru. Istnieją różne sposoby wyświetlania sformatowanej daty, a oto kilka pomocnych narzędzi dostarczanych przez Androida, aby to zrobić.

Utwórz listę opcji odbioru

Formater daty

Struktura systemu Android udostępnia klasę o nazwie SimpleDateFormat, która jest klasą do formatowania i analizowania dat w sposób zależny od ustawień regionalnych. Pozwala na formatowanie (data → tekst) i parsowanie (tekst → data) dat.

Możesz utworzyć instancję programu `SimpleDateFormat`, przekazując ciąg wzorca i ustawienia regionalne:

```
SimpleDateFormat("E MMM d", Locale.getDefault())
```

Ciąg wzorcowy, taki jak "E MMM d", jest reprezentacją formatów daty i godziny. Litery od 'A'do 'Z'i od 'a'do 'z'są interpretowane jako litery wzorca reprezentujące składniki ciągu daty lub godziny. Na przykład doznacza dzień w miesiącu, yrok i Mmiesiąc. Jeśli data to 4 stycznia 2018 r., ciąg wzorca "EEE, MMM d"jest analizowany do "Wed, Jul 4". Pełną listę wzorów liter można znaleźć w dokumentacji .
Obiekt Localereprezentuje określony region geograficzny, polityczny lub kulturowy. Reprezentuje kombinację języka/kraju/wariantu. Ustawienia regionalne służą do zmiany prezentacji informacji, takich jak liczby lub daty, w celu dostosowania do konwencji obowiązujących w regionie. Data i godzina są zależne od ustawień regionalnych, ponieważ są różnie zapisywane w różnych częściach świata. Użyjesz metody, Locale.getDefault()aby pobrać informacje o ustawieniach regionalnych ustawione na urządzeniu użytkownika i przekazać je do SimpleDateFormatkonstruktora.

Ustawienia regionalne w Androidzie to kombinacja kodu języka i kraju. Kody języków to dwuliterowe kody języków ISO, pisane małymi literami, takie jak „en" dla języka angielskiego. Kody krajów to dwuliterowe, pisane wielkimi literami kody krajów ISO, na przykład „US" dla Stanów Zjednoczonych.

Teraz użyj SimpleDateFormati Locale, aby określić dostępne daty odbioru aplikacji **Cupcake** .

1. W OrderViewModelklasie dodaj następującą funkcję wywoływaną getPickupOptions()do tworzenia i zwracania listy dat odbioru. W ramach metody utwórz valzmienną o nazwie optionsi zainicjuj ją w *.mutableListOf*<String>()

```
private fun getPickupOptions(): List<String> {
   val options = mutableListOf<String>()
}
```

2. Utwórz ciąg formatujący, używając SimpleDateFormatprzekazywania wzorca ciąg "E MMM d"i ustawień regionalnych. W ciągu wzorca Eoznacza nazwę dnia w tygodniu i jest analizowany jako „ **Wt Gru 10** ".

```
val formatter = SimpleDateFormat("E MMM d", Locale.getDefault())
```

   Importuj java.text.SimpleDateFormati java.util.Localepo wyświetleniu monitu przez Android Studio.

3. Pobierz Calendarinstancję i przypisz ją do nowej zmiennej. Niech to będzie val. Ta zmienna będzie zawierać aktualną datę i godzinę. Zaimportuj także java.util.Calendar.

```
val calendar = Calendar.getInstance()
```

4. Zbuduj listę dat, zaczynając od bieżącej daty i następujących trzech dat. Ponieważ będziesz potrzebować 4 opcji dat, powtórz ten blok kodu 4 razy. Ten repeatblok sformatuje datę, doda ją do listy opcji dat, a następnie zwiększy kalendarz o 1 dzień.

```
repeat(4) {
    options.add(formatter.format(calendar.time))
    calendar.add(Calendar.DATE, 1)
}
```

5. Zwróć zaktualizowane optionsna końcu metody. Oto Twoja ukończona metoda:

```
private fun getPickupOptions(): List<String> {
   val options = mutableListOf<String>()
   val formatter = SimpleDateFormat("E MMM d", Locale.getDefault())
```

```
  val calendar = Calendar.getInstance()
  // Create a list of dates starting with the current date and the following 3 dates
  repeat(4) {
     options.add(formatter.format(calendar.time))
     calendar.add(Calendar.DATE, 1)
  }
  return options
}
```

6. W OrderViewModelklasie dodaj właściwość klasy o nazwie dateOptionsthat's a val. Zainicjuj go za pomocą getPickupOptions()właśnie utworzonej metody.

   ```
   val dateOptions = getPickupOptions()
   ```

Zaktualizuj układ, aby wyświetlić opcje odbioru

Teraz, gdy masz już cztery dostępne daty odbioru w modelu widoku, zaktualizuj fragment_pickup.xmlukład, aby wyświetlał te daty. Użyjesz również powiązania danych, aby wyświetlić zaznaczony stan każdego przycisku opcji i zaktualizować datę w modelu widoku, gdy zostanie wybrany inny przycisk opcji. Ta implementacja jest podobna do wiązania danych we fragmencie smaku.

W fragment_pickup.xml:

Przycisk radiowy option0reprezentuje dateOptions[0]w viewModel(dzisiaj)

Przycisk radiowy option1reprezentuje dateOptions[1]w viewModel(jutro)

Przycisk radiowy option2reprezentuje dateOptions[2]w viewModel(pojutrze)

Przycisk radiowy option3reprezentuje dateOptions[3]za viewModel(dwa dni po jutrze)

1. W fragment_pickup.xmlprzypadku option0przycisku opcji użyj nowej zmiennej układu, viewModelaby ustawić checkedatrybut na podstawie datewartości w modelu widoku. Porównaj viewModel.datewłaściwość z pierwszym ciągiem na dateOptionsliście, który jest bieżącą datą. Użyj equalsfunkcji do porównania, a końcowe wyrażenie wiążące wygląda następująco:

   ```
   @{viewModel.date.equals(viewModel.dateOptions[0])}
   ```

2. Dla tego samego przycisku opcji dodaj detektor zdarzeń przy użyciu powiązania detektora z onClickatrybutem. Po kliknięciu tej opcji przycisku radiowego zadzwoń do setDate()on viewModel, przekazując dateOptions[0].
3. Dla tego samego przycisku opcji ustaw textwartość atrybutu na pierwszy ciąg na dateOptionsliście.

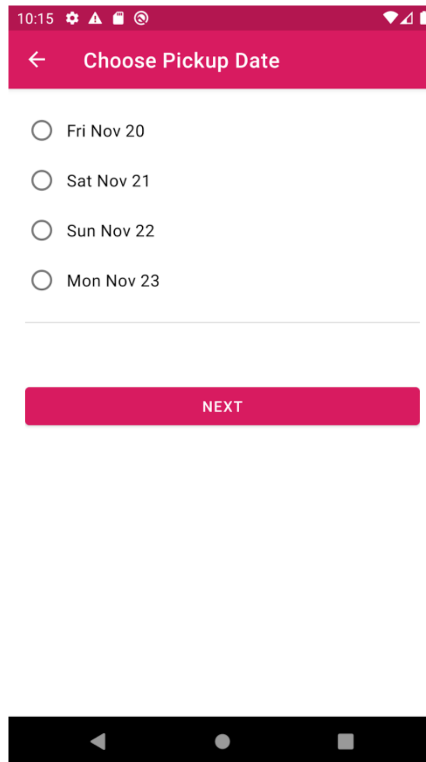
   ```
   <RadioButton
     android:id="@+id/option0"
     ...
     android:checked="@{viewModel.date.equals(viewModel.dateOptions[0])}"
     android:onClick="@{() -> viewModel.setDate(viewModel.dateOptions[0])}"
     android:text="@{viewModel.dateOptions[0]}"
     ...
   ```

```
/>
```

4. Powtórz powyższe kroki dla innych przycisków radiowych, odpowiednio zmień indeks `dateOptions`.

```
<RadioButton
  android:id="@+id/option1"
  ...
  android:checked="@{viewModel.date.equals(viewModel.dateOptions[1])}"
  android:onClick="@{() -> viewModel.setDate(viewModel.dateOptions[1])}"
  android:text="@{viewModel.dateOptions[1]}"
  ... />

<RadioButton
  android:id="@+id/option2"
  ...
  android:checked="@{viewModel.date.equals(viewModel.dateOptions[2])}"
  android:onClick="@{() -> viewModel.setDate(viewModel.dateOptions[2])}"
  android:text="@{viewModel.dateOptions[2]}"
  ... />

<RadioButton
  android:id="@+id/option3"
  ...
  android:checked="@{viewModel.date.equals(viewModel.dateOptions[3])}"
  android:onClick="@{() -> viewModel.setDate(viewModel.dateOptions[3])}"
  android:text="@{viewModel.dateOptions[3]}"
  ... />
```

5. Uruchom aplikację i powinieneś zobaczyć dostępne opcje odbioru w ciągu najbliższych kilku dni. Twój zrzut ekranu będzie się różnić w zależności od tego, jaki jest dla Ciebie bieżący dzień. Zauważ, że domyślnie nie jest wybrana żadna opcja. Wdrożysz to w następnym kroku.



6. W OrderViewModelklasie utwórz funkcję o nazwie resetOrder(), aby zresetować MutableLiveDatawłaściwości w modelu widoku. Przypisz bieżącą wartość daty z dateOptionslisty do _date.*value.*

```
fun resetOrder() {
  _quantity.value = 0
  _flavor.value = ""
  _date.value = dateOptions[0]
  _price.value = 0.0
}
```

7. Dodaj initblok do klasy i wywołaj z niego nową metodę resetOrder().

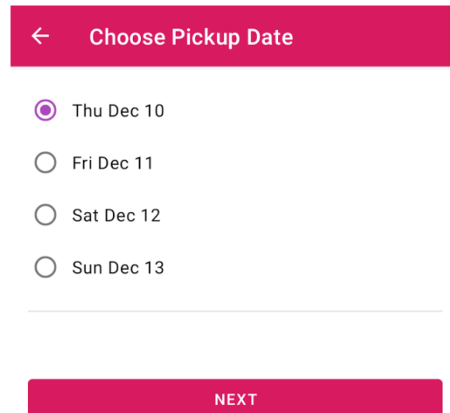
```
init {
  resetOrder()
}
```

8. Usuń początkowe wartości z deklaracji właściwości w klasie. Teraz używasz initbloku do inicjowania właściwości podczas tworzenia wystąpienia OrderViewModel.

```
private val _quantity = MutableLiveData<Int>()
val quantity: LiveData<Int> = _quantity

private val _flavor = MutableLiveData<String>()
val flavor: LiveData<String> = _flavor

private val _date = MutableLiveData<String>()
val date: LiveData<String> = _date
```

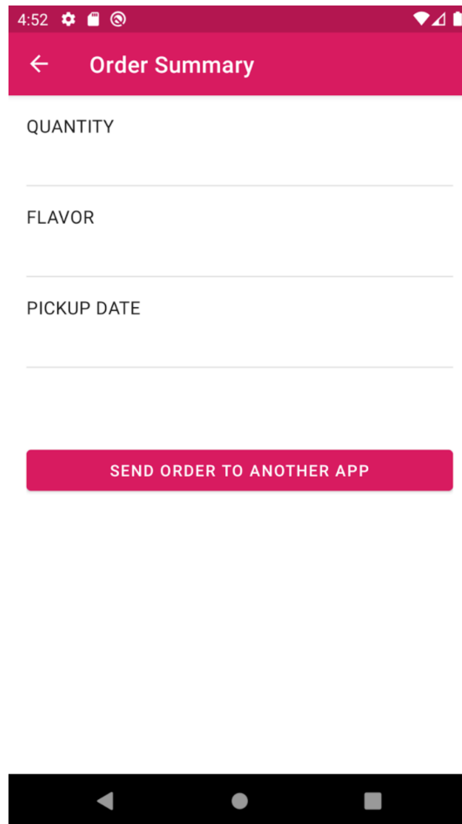
```
private val _price = MutableLiveData<Double>()
val price: LiveData<Double> = _price
```

9. Uruchom aplikację ponownie, zwróć uwagę, że dzisiejsza data jest domyślnie wybrana.



Zaktualizuj fragment podsumowania, aby użyć modelu widoku

Przejdźmy teraz do ostatniego fragmentu. Fragment podsumowania zamówienia ma na celu pokazanie podsumowania szczegółów zamówienia. W tym zadaniu wykorzystujesz wszystkie informacje o zamówieniu z modelu widoku współdzielonego i aktualizujesz szczegóły zamówienia na ekranie za pomocą powiązania danych.



1. W fragment_summary.xmlprogramie upewnij się, że masz viewModelzadeklarowaną zmienną danych modelu widoku.

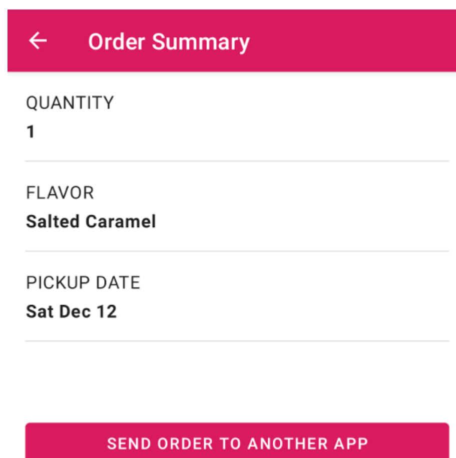
```
<layout ...>

  <data>
    <variable
      name="viewModel"
      type="com.example.cupcake.model.OrderViewModel" />
  </data>

  <ScrollView ...>

  ...
```

2. W SummaryFragment, w onViewCreated(), upewnij się, że binding.viewModeljest zainicjowany.
3. W fragment_summary.xmlprogramie odczytaj model widoku, aby zaktualizować ekran szczegółami podsumowania zamówienia. Zaktualizuj ilość, smak i datę TextViews, dodając następujące atrybuty tekstowe. Ilość jest typu Int, więc musisz przekonwertować ją na łańcuch.

```
<TextView
  android:id="@+id/quantity"
  ...
  android:text="@{viewModel.quantity.toString()}"
  ... />
```

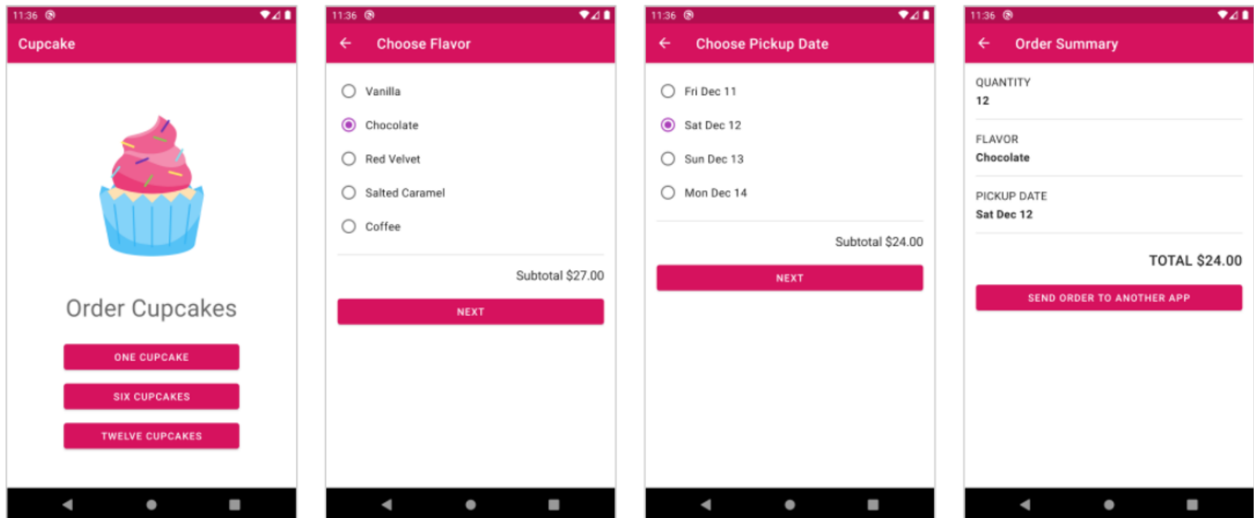
```
<TextView
  android:id="@+id/flavor"
  ...
  android:text="@{viewModel.flavor}"
  ... />

<TextView
  android:id="@+id/date"
  ...
  android:text="@{viewModel.date}"
  ... />
```

4. Uruchom i przetestuj aplikację, aby sprawdzić, czy wybrane opcje zamówienia pojawiają się w podsumowaniu zamówienia.



8. Oblicz cenę ze szczegółów zamówienia

Patrząc na ostatnie zrzuty ekranu aplikacji z tego ćwiczenia kodowania, zauważysz, że cena jest faktycznie wyświetlana na każdym fragmencie (z wyjątkiem `StartFragment`), więc użytkownik zna cenę podczas tworzenia zamówienia.



Oto zasady naszego sklepu z babeczkami dotyczące obliczania ceny.

- Każda babeczka kosztuje 2,00 USD!
- Odbiór tego samego dnia dodaje dodatkowe 3,00 USD do zamówienia

Zatem przy zamówieniu 6 babeczek cena wyniesie 6 babeczek x 2 USD za sztukę = 12 USD. Jeśli użytkownik chce odebrać przesyłkę tego samego dnia, dodatkowy koszt w wysokości 3 USD spowoduje, że łączna cena zamówienia wyniesie 15 USD.

Zaktualizuj cenę w widoku modelu

Aby dodać obsługę tej funkcji w swojej aplikacji, najpierw zajmij się ceną za babeczkę i na razie zignoruj koszt odbioru tego samego dnia.

1. Otwórz `OrderViewModel.kt` i zapisz cenę za babeczkę w zmiennej. Zadeklaruj ją jako stałą prywatną najwyższego poziomu na górze pliku, poza definicją klasy (ale po instrukcjach import). Użyj `const` modyfikatora i ustaw go tylko do odczytu use `val`.

```
package ...

import ...

private const val PRICE_PER_CUPCAKE = 2.00

class OrderViewModel : ViewModel() {
    ...
```

   Przypomnij sobie, że wartości stałe (oznaczone `const` słowem kluczowym w Kotlinie) nie zmieniają się, a wartość jest znana w czasie kompilacji. Aby dowiedzieć się więcej o stałych, zapoznaj się z dokumentacją .

2. Teraz, gdy już zdefiniowałeś cenę za babeczkę, utwórz pomocniczą metodę obliczania ceny. Ta metoda może być `private` spowodowana tym, że jest używana tylko w tej klasie. Zmienisz logikę ceny, aby w następnym zadaniu uwzględniać opłaty za odbiór tego samego dnia.

```
private fun updatePrice() {
    _price.value = (quantity.value ?: 0) * PRICE_PER_CUPCAKE
```

```
}
```

Ta linia kodu mnoży cenę za babeczkę przez ilość zamówionych babeczek. W przypadku kodu w nawiasach, ponieważ wartość quantity.valuemoże być null, użyj operatora elvisa (?:). Operator elvisa (?:) oznacza, że jeśli wyrażenie po lewej stronie nie jest puste, użyj go. W przeciwnym razie, jeśli wyrażenie po lewej stronie ma wartość null, użyj wyrażenia po prawej stronie operatora elvis (0w tym przypadku).

Ciekawostka: operator Elvisa (?:) nosi imię gwiazdy rocka, Elvisa Presleya, ponieważ gdy patrzysz na niego z boku, przypomina emotikon Elvisa Presleya z jego quiffem .

3. W tej samej OrderViewModelklasie zaktualizuj zmienną ceny po ustawieniu ilości. Wywołaj nową funkcję w setQuantity()funkcji.

```
fun setQuantity(numberCupcakes: Int) {
   _quantity.value = numberCupcakes
   updatePrice()
}
```

Powiąż właściwość ceny z interfejsem użytkownika

1. W układach dla fragment_flavor.xml, fragment_pickup.xmli fragment_summary.xml, upewnij się, że zmienna danych viewModeltypu com.example.cupcake.model.OrderViewModeljest zdefiniowana.

```
<layout ...>

   <data>
      <variable
         name="viewModel"
         type="com.example.cupcake.model.OrderViewModel" />
   </data>

   <ScrollView ...>

   ...
```

2. W onViewCreated()metodzie każdej klasy fragmentu upewnij się, że powiązałeś instancję obiektu widoku modelu we fragmencie ze zmienną danych modelu widoku w układzie.

```
binding?.apply {
   viewModel = sharedViewModel
   ...
}
```

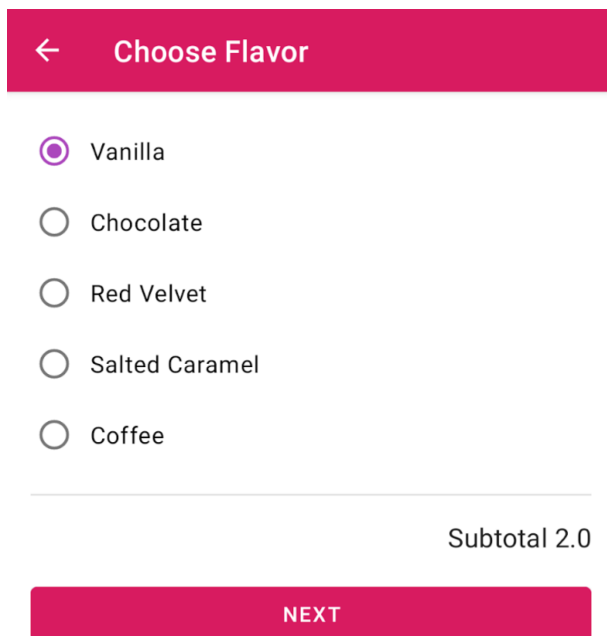
3. W każdym układzie fragmentu użyj viewModelzmiennej, aby ustawić cenę, jeśli jest pokazana w układzie. Zacznij od modyfikacji fragment_flavor.xmlpliku. Dla subtotalwidoku tekstowego ustaw wartość android:textatrybutu na "@{@string/subtotal_price(viewModel.price)}".To wyrażenie układu powiązania danych używa zasobu ciągu @string/subtotal_pricei przekazuje parametr, który jest ceną z modelu widoku, więc dane wyjściowe będą zawierać na przykład **sumę częściową 12.0 .**

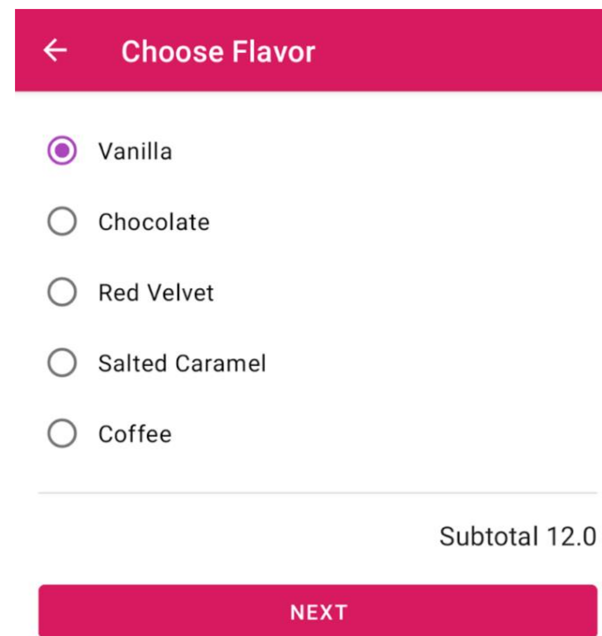
```
...

<TextView
    android:id="@+id/subtotal"
    android:text="@{@string/subtotal_price(viewModel.price)}"
    ... />

...
```

Używasz tego zasobu tekstowego, który został już zadeklarowany w `strings.xml`pliku:

```
<string name="subtotal_price">Subtotal %s</string>
```

4. Uruchom aplikację. Jeśli wybierzesz **Jedna babeczka** we fragmencie początkowym, fragment smaku pokaże **Suma częściowa 2.0** . Jeśli wybierzesz **Sześć babeczek** , fragment smaku pokaże **Suma częściowa 12.0** itd. Później sformatujesz cenę we właściwym formacie waluty, więc takie zachowanie jest na razie oczekiwane.





5. Teraz dokonaj podobnej zmiany dla fragmentów odbioru i podsumowania. W układach `fragment_pickup.xml`i `fragment_summary.xml`zmodyfikuj widoki tekstowe, aby również korzystać z `viewModel` `price`właściwości.

`fragment_pickup.xml`

```
...

<TextView
    android:id="@+id/subtotal"
    ...
    android:text="@{@string/subtotal_price(viewModel.price)}"
    ... />

...
```

`fragment_summary.xml`

```
...

<TextView
   android:id="@+id/total"
   ...
   android:text="@{@string/total_price(viewModel.price)}"
   ... />


...
```

4. Uruchom aplikację. Upewnij się, że cena pokazana w podsumowaniu zamówienia jest poprawnie obliczona dla ilości zamówienia 1, 6 i 12 babeczek. Jak wspomniano, oczekuje się, że formatowanie ceny nie jest w tej chwili poprawne (wyświetla 2,0 za 2 USD lub 12,0 za 12 USD).

Choose Pickup Date

Thu Dec 10

Fri Dec 11

Sat Dec 12

Sun Dec 13

Subtotal 12.0

NEXT

Order Summary

QUANTITY

6

FLAVOR

Vanilla

PICKUP DATE

Thu Dec 10

TOTAL 12.0

SEND ORDER TO ANOTHER APP

Dopłata za odbiór tego samego dnia

W tym zadaniu wdrożysz drugą zasadę, która mówi, że odbiór tego samego dnia dodaje do zamówienia dodatkowe 3,00 USD.

1. Na OrderViewModelzajęciach zdefiniuj nową stałą prywatną najwyższego poziomu dla tego samego dnia kosztu odbioru.

```
private const val PRICE_FOR_SAME_DAY_PICKUP = 3.00
```

2. W updatePrice(), sprawdź, czy użytkownik wybrał odbiór tego samego dnia. Sprawdź, czy data w modelu widoku (_date.*value*) jest taka sama jak pierwsza pozycja na dateOptionsliście, która zawsze jest dniem bieżącym.

```
private fun updatePrice() {
    _price.value = (quantity.value ?: 0) * PRICE_PER_CUPCAKE
    if (dateOptions[0] == _date.value) {

    }
```

```
}
```

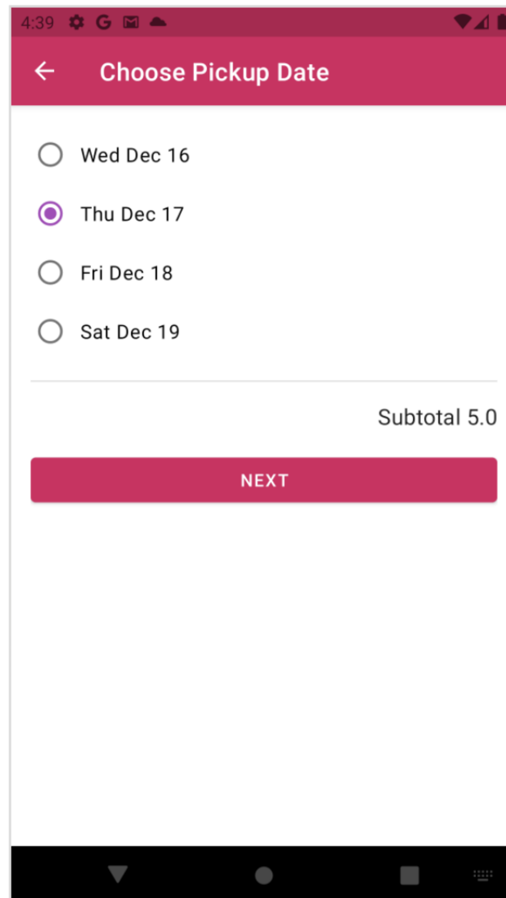
3. Aby uprościć te obliczenia, wprowadź zmienną tymczasową, calculatedPrice. Oblicz zaktualizowaną cenę i przypisz ją z powrotem do _price.*value*.

```
private fun updatePrice() {
    var calculatedPrice = (quantity.value ?: 0) * PRICE_PER_CUPCAKE
    // If the user selected the first option (today) for pickup, add the surcharge
    if (dateOptions[0] == _date.value) {
        calculatedPrice += PRICE_FOR_SAME_DAY_PICKUP
    }
    _price.value = calculatedPrice
}
```

4. Zadzwoń do updatePrice()metody pomocnika z setDate()metody, aby dodać opłaty za odbiór tego samego dnia.

```
fun setDate(pickupDate: String) {
    _date.value = pickupDate
    updatePrice()
}
```

5. Uruchom swoją aplikację, poruszaj się po aplikacji. Zauważysz, że zmiana daty odbioru nie usuwa opłat za odbiór tego samego dnia z całkowitej ceny. Dzieje się tak, ponieważ cena jest zmieniana w modelu widoku, ale nie jest powiadamiana o wiążącym układzie.



Ustaw właściciela cyklu życia, aby obserwował LiveData

LifecycleOwnerto klasa, która ma cykl życia Androida, taka jak aktywność lub fragment. Obserwator obserwuje zmiany w danych aplikacji tylko wtedy LiveData, gdy właściciel cyklu życia jest w stanie aktywnym (STARTEDlub RESUMED).

W Twojej aplikacji LiveDataobiekt lub obserwowalne dane są pricewłaściwością w modelu widoku. Właścicielami cyklu życia są fragmenty smaku, odbioru i podsumowania. Obserwatorzy są LiveDatawyrażeniami wiążącymi w plikach układu z możliwymi do zaobserwowania danymi, takimi jak cena. Dzięki powiązaniu danych, gdy obserwowalna wartość ulegnie zmianie, elementy interfejsu użytkownika, z którymi jest powiązany, są automatycznie aktualizowane.

Przykład wyrażenia wiążącego: `android:text="@{@string/subtotal_price(viewModel.price)}"`

Aby elementy interfejsu użytkownika aktualizowały się automatycznie, musisz skojarzyć`binding.lifecycleOwner`

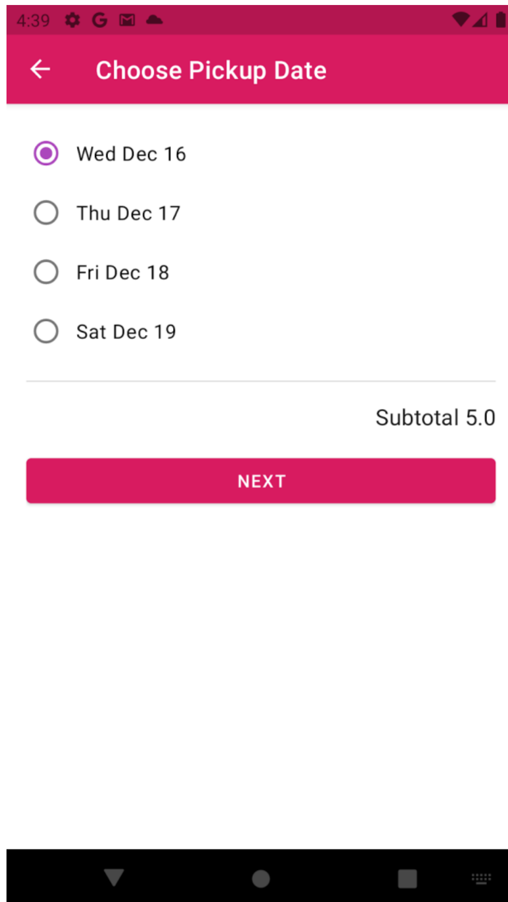
z właścicielami cyklu życia w aplikacji. Wdrożysz to później.

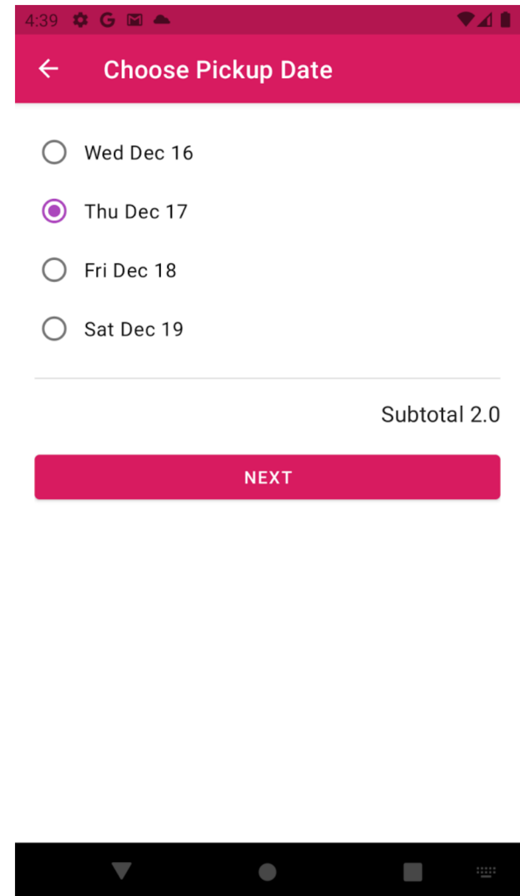
1. W klasach FlavorFragment, PickupFragment, SummaryFragmentwewnątrz onViewCreated()metody dodaj w binding?.applybloku następujące elementy. Spowoduje to ustawienie właściciela cyklu życia w obiekcie powiązania. Ustawiając właściciela cyklu życia, aplikacja będzie mogła obserwować LiveDataobiekty.

```
binding?.apply {
    lifecycleOwner = viewLifecycleOwner
    ...
```

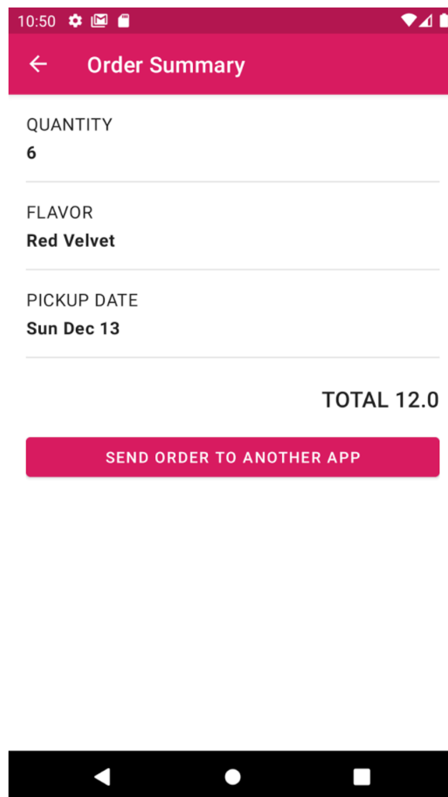
}

2. Uruchom ponownie swoją aplikację. Na ekranie odbioru zmień datę odbioru i zwróć uwagę na różnicę w automatycznej zmianie ceny. A opłaty za odbiór są prawidłowo odzwierciedlone na ekranie podsumowania.
3. Zwróć uwagę, że po wybraniu dzisiejszej daty odbioru cena zamówienia wzrasta o 3,00 USD. Cena za wybór przyszłej daty powinna nadal wynosić ilość babeczek x 2,00 USD.





4. Przetestuj różne przypadki z różnymi ilościami ciastek, smakami i datami odbioru. Teraz powinieneś zobaczyć aktualizację ceny z modelu widoku na każdym fragmencie. Najlepsze jest to, że nie trzeba było pisać dodatkowego kodu Kotlin, aby za każdym razem aktualizować interfejs użytkownika o cenę.



Aby dokończyć wdrażanie funkcji ceny, musisz sformatować cenę na walutę lokalną.

Sformatuj cenę za pomocą transformacji LiveData

Metody LiveDatatransformacji zapewniają sposób wykonywania manipulacji danymi na źródle LiveDatai zwracania wynikowego LiveDataobiektu. Mówiąc prościej, przekształca wartość LiveDataw inną wartość. Te przekształcenia nie są obliczane, chyba że obserwator obserwuje LiveDataobiekt.

Jest Transformations.map()to jedna z funkcji transformacji, ta metoda przyjmuje źródło LiveDatai funkcję jako parametry. Funkcja manipuluje źródłem LiveDatai zwraca zaktualizowaną wartość, którą również można zaobserwować.

Kilka przykładów w czasie rzeczywistym, w których możesz użyć transformacji LiveData:

- Formatuj ciągi daty i godziny do wyświetlenia
- Sortowanie listy przedmiotów
- Filtrowanie lub grupowanie elementów
- Oblicz wynik z listy, na przykład sumę wszystkich elementów, liczbę elementów, zwróć ostatni element i tak dalej.

W tym zadaniu użyjesz Transformations.map()metody formatowania ceny w walucie lokalnej. Przekształcisz pierwotną cenę z wartości dziesiętnej (LiveData<Double>) na wartość ciągu (LiveData<String>).

1. W OrderViewModelklasie zmień typ właściwości podkładu na LiveData<String>zamiast LiveData<Double>.Sformatowana cena będzie ciągiem z symbolem waluty, takim jak „$”. W następnym kroku naprawisz błąd inicjalizacji.

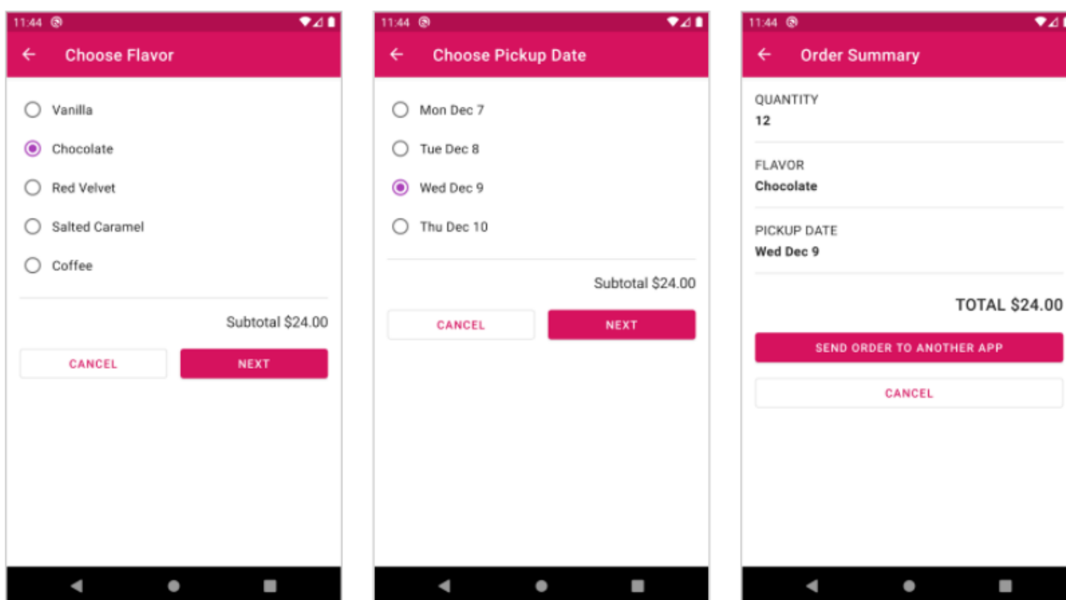
```
private val _price = MutableLiveData<Double>()
val price: LiveData<String>
```

2. Użyj Transformations.map(), aby zainicjować nową zmienną, przekaż _pricefunkcję lambda i . Użyj getCurrencyInstance()metody w NumberFormatklasie, aby przekonwertować cenę na format w walucie lokalnej. Kod transformacji będzie wyglądał tak.

```
private val _price = MutableLiveData<Double>()
val price: LiveData<String> = Transformations.map(_price) {
   NumberFormat.getCurrencyInstance().format(it)
}
```

Musisz zaimportować androidx.lifecycle.Transformationsi java.text.NumberFormat.

3. Uruchom aplikację. Teraz powinieneś zobaczyć sformatowany ciąg ceny dla sumy częściowej i sumy. Jest to o wiele bardziej przyjazne dla użytkownika!



4. Sprawdź, czy działa zgodnie z oczekiwaniami. Przypadki testowe takie jak: Zamów jedną babeczkę, zamów sześć babeczek, zamów 12 babeczek. Upewnij się, że cena jest poprawnie aktualizowana na każdym ekranie. Powinien zawierać **sumę częściową 2,00 USD** za fragmenty smaku i odbioru oraz **sumę 2,00 USD** za podsumowanie zamówienia. Upewnij się również, że podsumowanie zamówienia zawiera prawidłowe szczegóły zamówienia.

9. Skonfiguruj odbiorniki kliknięcia za pomocą wiązania słuchacza

W tym zadaniu użyjesz wiązania słuchaczy, aby powiązać detektory kliknięcia przycisku w klasach fragmentów z układem.

1. W pliku układu fragment_start.xmldodaj zmienną danych o nazwie startFragmenttypu com.example.cupcake.StartFragment. Upewnij się, że nazwa pakietu fragmentu jest zgodna z nazwą pakietu Twojej aplikacji.

```
<layout ...>

   <data>
      <variable
         name="startFragment"
         type="com.example.cupcake.StartFragment" />
   </data>
   ...
   <ScrollView ...>
```

2. W metodzie StartFragment.ktw onViewCreated()metodzie powiąż nową zmienną danych z wystąpieniem fragmentu. Możesz uzyskać dostęp do instancji fragmentu wewnątrz fragmentu za pomocą thissłowa kluczowego. Usuń binding?.*apply*blok i wraz z zawartym w nim kodem. Ukończona metoda powinna wyglądać tak.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
   super.onViewCreated(view, savedInstanceState)
   binding?.startFragment = this
}
```

3. W fragment_start.xmlprogramie dodaj detektory zdarzeń za pomocą powiązania detektora z onClickatrybutem przycisków, wykonaj wywołanie orderCupcake()on startFragment, przekazując liczbę babeczek.

```
<Button
   android:id="@+id/order_one_cupcake"
   android:onClick="@{() -> startFragment.orderCupcake(1)}"
   ... />

<Button
   android:id="@+id/order_six_cupcakes"
   android:onClick="@{() -> startFragment.orderCupcake(6)}"
   ... />

<Button
   android:id="@+id/order_twelve_cupcakes"
   android:onClick="@{() -> startFragment.orderCupcake(12)}"
   ... />
```

4. Uruchom aplikację. Zwróć uwagę, że obsługa kliknięcia przycisku we fragmencie początkowym działa zgodnie z oczekiwaniami.
5. Podobnie dodaj powyższą zmienną danych w innych układach, aby powiązać wystąpienie fragmentu fragment_flavor.xml, fragment_pickup.xmli fragment_summary.xml.

Wfragment_flavor.xml

```
<layout ...>

   <data>
```

```
        <variable
            ... />

        <variable
            name="flavorFragment"
            type="com.example.cupcake.FlavorFragment" />
    </data>

    <ScrollView ...>
```

W `fragment_pickup.xml`:

```
<layout ...>

    <data>
        <variable
            ... />

        <variable
            name="pickupFragment"
            type="com.example.cupcake.PickupFragment" />
    </data>

    <ScrollView ...>
```

W `fragment_summary.xml`:

```
<layout ...>

    <data>
        <variable
            ... />

        <variable
            name="summaryFragment"
            type="com.example.cupcake.SummaryFragment" />
    </data>

    <ScrollView ...>
```

6. W pozostałych klasach fragmentów w `onViewCreated()`metodach usuń kod, który ręcznie ustawia detektor kliknięć na przyciskach.
7. W `onViewCreated()`metodach powiąż zmienną danych fragmentu z instancją fragmentu. Tutaj użyjesz `this`słowa kluczowego inaczej, ponieważ wewnątrz `binding?.apply`bloku słowo kluczowe `this`odnosi się do instancji wiążącej, a nie instancji fragmentu. Użyj `@`i wyraźnie określ

nazwę klasy fragmentu, na przykład this@FlavorFragment. Wypełnione onViewCreated()metody powinny wyglądać następująco:

Metoda onViewCreated()w FlavorFragmentklasie powinna wyglądać tak:

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

    binding?.apply {
        lifecycleOwner = viewLifecycleOwner
        viewModel = sharedViewModel
        flavorFragment = this@FlavorFragment
    }
}
```

Metoda onViewCreated()w PickupFragmentklasie powinna wyglądać tak:

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

   binding?.apply {
       lifecycleOwner = viewLifecycleOwner
       viewModel = sharedViewModel
       pickupFragment = this@PickupFragment
   }
}
```

Wynikowa onViewCreated()metoda w metodzie SummaryFragmentklasy powinna wyglądać tak:

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

   binding?.apply {
       lifecycleOwner = viewLifecycleOwner
       viewModel = sharedViewModel
       summaryFragment = this@SummaryFragment
   }
}
```

8. Podobnie w innych plikach układu Dodaj wyrażenia powiązania odbiornika do onClickatrybutu przycisków.

   W fragment_flavor.xml:

```
<Button
    android:id="@+id/next_button"
    android:onClick="@{() -> flavorFragment.goToNextScreen()}"
```

```
    ... />
```

W `fragment_pickup.xml`:

```
<Button
    android:id="@+id/next_button"
    android:onClick="@{() -> pickupFragment.goToNextScreen()}"
    ... />
```

W `fragment_summary.xml`:

```
<Button
    android:id="@+id/send_button"
    android:onClick="@{() -> summaryFragment.sendOrder()}"
    ...>
```

9. Uruchom aplikację, aby sprawdzić, czy przyciski nadal działają zgodnie z oczekiwaniami. Nie powinno być żadnych widocznych zmian w zachowaniu, ale teraz użyłeś powiązań detektorów do skonfigurowania detektorów kliknięć!

   Gratulujemy ukończenia tego ćwiczenia z programowania i stworzenia aplikacji **Cupcake** ! Jednak aplikacja nie jest jeszcze gotowa. W następnym ćwiczeniu z programowania dodasz przycisk **Anuluj** i zmodyfikujesz stos. Dowiesz się również, co to jest backstack i inne nowe tematy. Do zobaczenia tam!

10. Kod rozwiązania

Kod rozwiązania dla tego ćwiczenia programowania znajduje się w projekcie pokazanym poniżej. Użyj gałęzi viewmodel, aby pobrać lub pobrać kod.

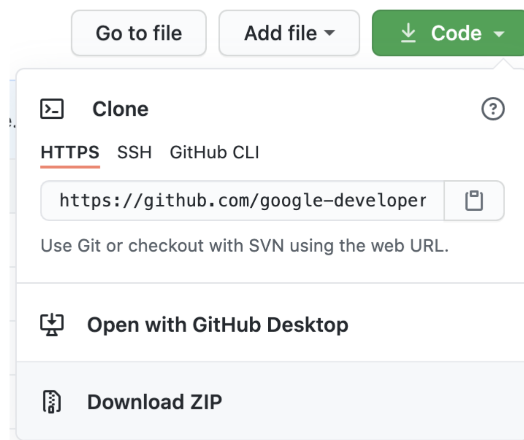
Adres URL kodu rozwiązania:

https://github.com/google-developer-training/android-basics-kotlin-cupcake-app/tree/viewmodel

Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

Pobierz kod

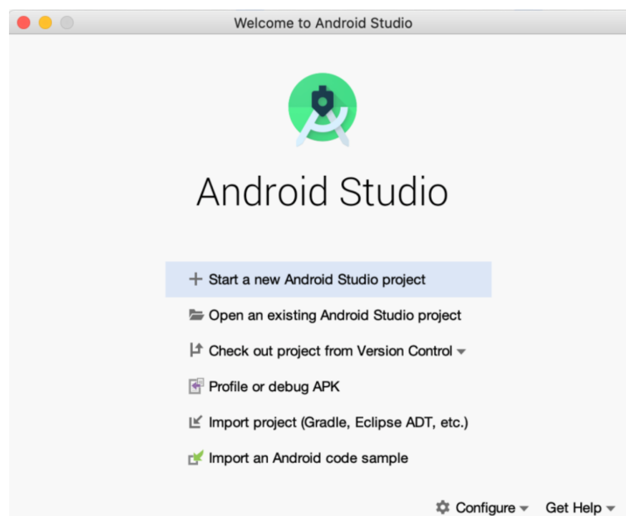
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



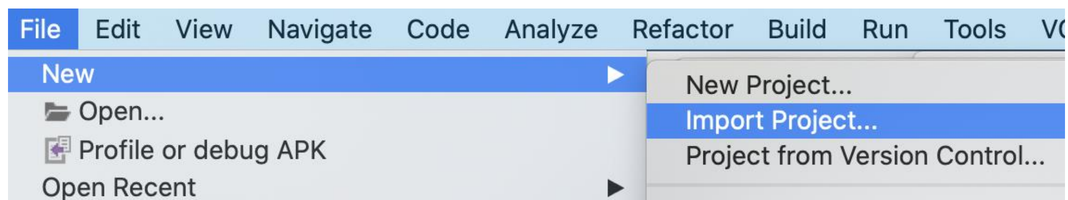
3. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
4. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
5. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

Otwórz projekt w Android Studio

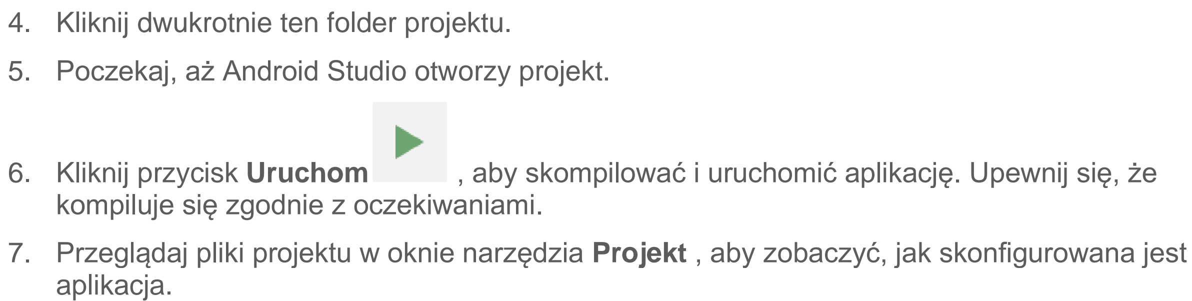
1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz istniejący projekt Android Studio.**



Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Nowy** > **Importuj projekt** .



3. W oknie dialogowym **Importuj projekt** przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).

4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.
6. Kliknij przycisk **Uruchom**  , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.
7. Przeglądaj pliki projektu w oknie narzędzia **Projekt** , aby zobaczyć, jak skonfigurowana jest aplikacja.

11. Podsumowanie

- Jest ViewModelczęścią komponentów architektury Androida , a dane aplikacji zapisane w ramach ViewModelsą zachowywane podczas zmian konfiguracji. Aby dodać a ViewModeldo swojej aplikacji, tworzysz nowe zajęcia i rozszerzasz je z ViewModelzajęć.
- SharedViewModel służy do zapisywania danych aplikacji z wielu fragmentów w jednym ViewModel. Wiele fragmentów w aplikacji będzie uzyskiwać dostęp do udostępnionego ViewModelprzy użyciu ich zakresu aktywności.
- LifecycleOwnerto klasa, która ma cykl życia Androida, taka jak aktywność lub fragment.
- LiveDataobserwator obserwuje zmiany w danych aplikacji tylko wtedy, gdy właściciel cyklu życia jest w stanie aktywnym (STARTEDlub RESUMED).
- Powiązania odbiornika to wyrażenia lambda, które są uruchamiane, gdy wystąpi zdarzenie, takie jak onClickzdarzenie. Są one podobne do odwołań do metod, na przykład textview.setOnClickListener(clickListener)powiązania odbiornika umożliwiają uruchamianie dowolnych wyrażeń powiązania danych.
- Metody LiveDatatransformacji zapewniają sposób wykonywania manipulacji danymi na źródle LiveDatai zwracania wynikowego LiveDataobiektu.
- Platformy Android udostępniają klasę o nazwie SimpleDateFormat, klasę do formatowania i analizowania dat w sposób zależny od ustawień regionalnych. Pozwala na formatowanie (data → tekst) i parsowanie (tekst → data) dat.

12. Dowiedz się więcej

- Komponent nawigacji
- ZobaczPrzegląd Modelu
- Wiązanie danych
- Układ i wiązania wyrażeń
- Przekształć dane na żywo
- Prosty format daty
- applyfunkcja scope w Kotlin
- Stałe czasu kompilacji

Nawigacja i tylny stos

1. Zanim zaczniesz

W tym ćwiczeniu z programowania zakończysz implementację pozostałej części aplikacji **Cupcake** , którą rozpocząłeś w poprzednim ćwiczeniu z programowania. Aplikacja **Cupcake** ma wiele ekranów i pokazuje przepływ zamówień babeczek. Ukończona aplikacja powinna umożliwiać użytkownikowi poruszanie się po aplikacji w celu:

- Utwórz zamówienie na babeczki
- Użyj przycisku **W górę** lub **Wstecz** , aby przejść do poprzedniego kroku przepływu zamówienia
- Anuluj zamówienie
- Wyślij zamówienie do innej aplikacji, takiej jak aplikacja do obsługi poczty e-mail

Po drodze dowiesz się, jak Android obsługuje zadania i tylny stos aplikacji. Umożliwi to manipulowanie stosem tylnym w scenariuszach, takich jak anulowanie zamówienia, co powoduje powrót użytkownika do pierwszego ekranu aplikacji (w przeciwieństwie do poprzedniego ekranu przepływu zamówienia).

Warunki wstępne

- Potrafi tworzyć i używać współdzielonego modelu widoku we fragmentach działania
- Zaznajomiony z używaniem komponentu Jetpack Navigation
- Użyto wiązania danych z LiveData, aby zachować synchronizację interfejsu użytkownika z modelem widoku
- Potrafi zbudować zamiar rozpoczęcia nowej działalności

Czego się nauczysz

- Jak nawigacja wpływa na tylny stos aplikacji
- Jak zaimplementować niestandardowe zachowanie stosu wstecznego

Co zbudujesz

- Aplikacja do zamawiania babeczek, która umożliwia wysłanie zamówienia do innej aplikacji i anulowanie zamówienia

Czego potrzebujesz

- Komputer z zainstalowanym Android Studio.
- Kod aplikacji **Cupcake** z poprzedniego ćwiczenia z programowania

2. Przegląd aplikacji startowej

To laboratorium programowania korzysta z aplikacji **Cupcake** z poprzedniego ćwiczenia z programowania. Możesz użyć kodu z poprzedniego ćwiczenia z programowania lub pobrać kod startowy z GitHub.

Pobierz kod startowy do tego ćwiczenia z programowania

Jeśli pobierzesz kod startowy z GitHub, zwróć uwagę, że nazwa folderu projektu to `android-basics-kotlin-cupcake-app-viewmodel`. Wybierz ten folder podczas otwierania projektu w Android Studio.

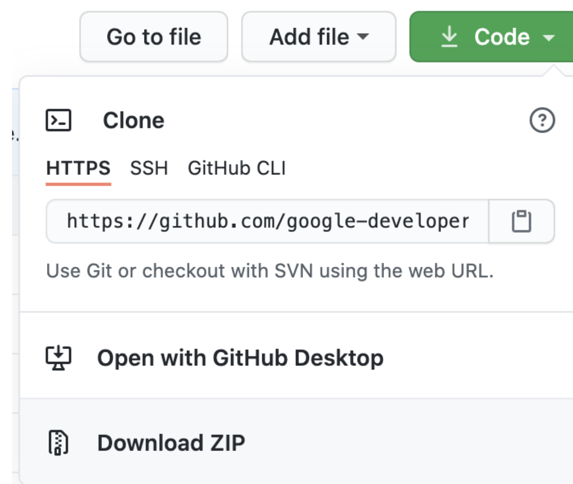
Adres URL kodu startowego:

https://github.com/google-developer-training/android-basics-kotlin-cupcake-app/tree/viewmodel

Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

Pobierz kod

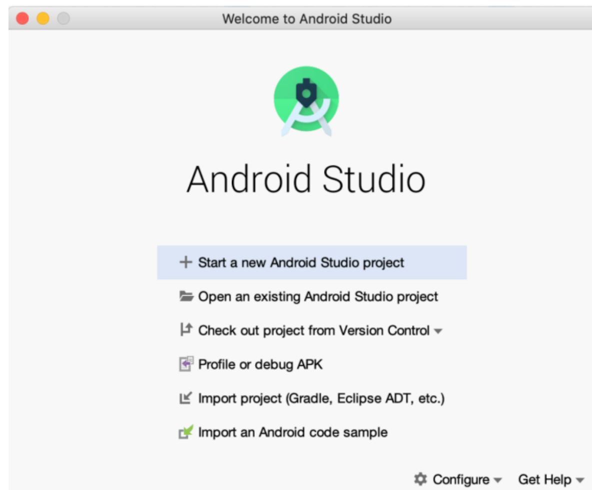
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



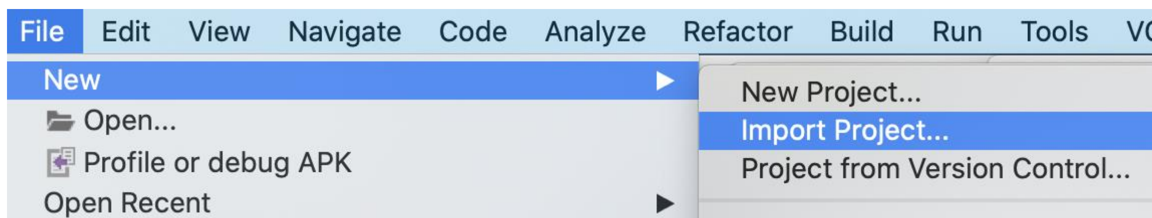
3. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
4. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
5. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

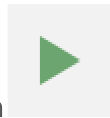
Otwórz projekt w Android Studio

1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz istniejący projekt Android Studio.**

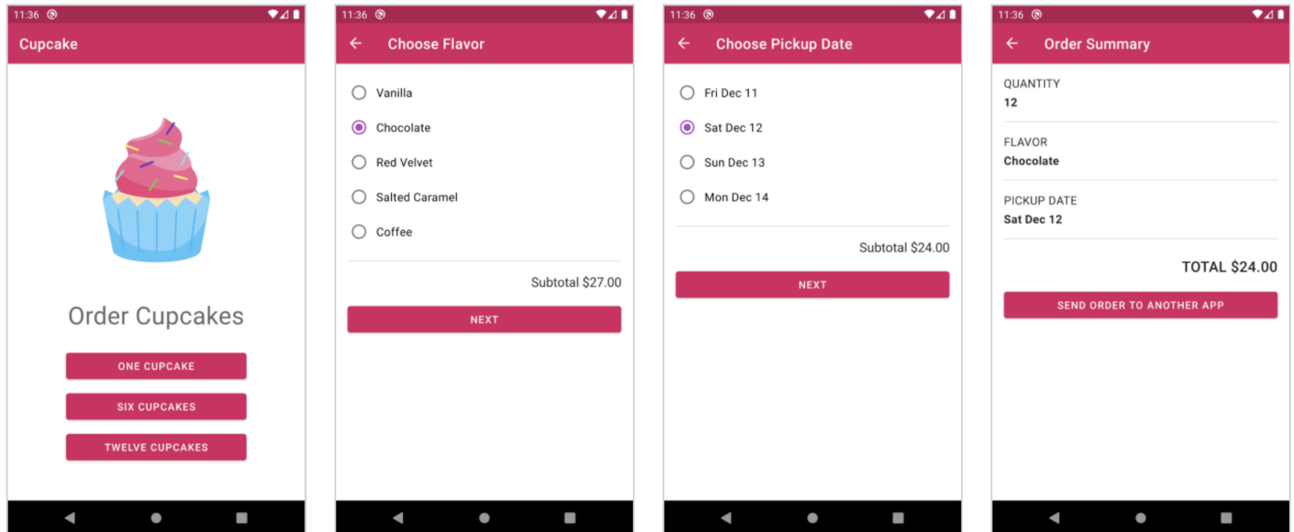


Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Nowy** > **Importuj projekt** .

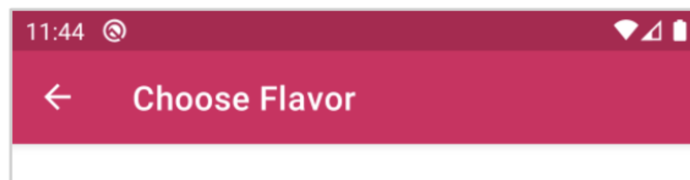


3. W oknie dialogowym **Importuj projekt** przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.
6. Kliknij przycisk **Uruchom**  , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.
7. Przeglądaj pliki projektu w oknie narzędzia **Projekt** , aby zobaczyć, jak skonfigurowana jest aplikacja.

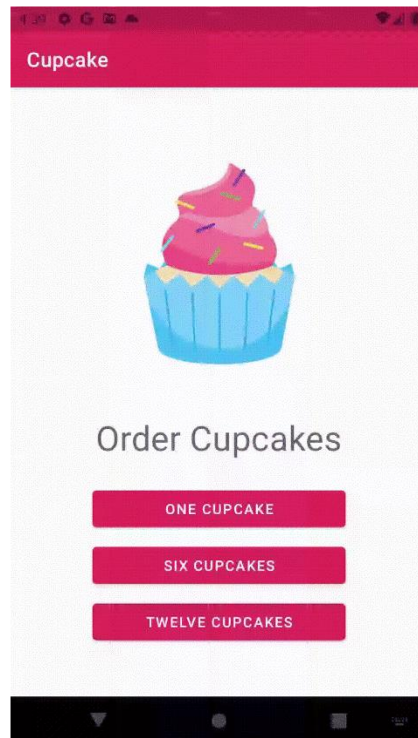
Teraz uruchom aplikację i powinna wyglądać tak.



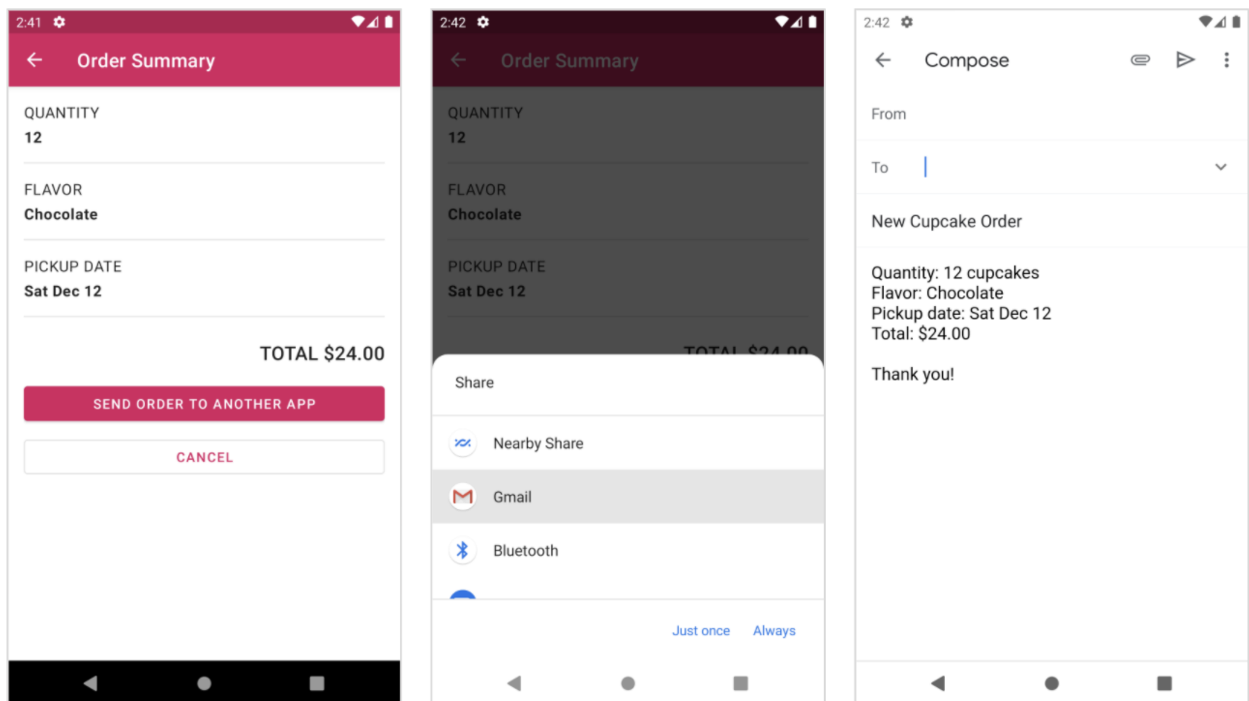
W tym ćwiczeniu z programowania najpierw zakończysz implementację przycisku **W górę** w aplikacji, aby dotknięcie go przeniosło użytkownika do poprzedniego kroku przepływu zamówienia.



Następnie zostanie dodany przycisk **Anuluj** , aby użytkownik mógł anulować zamówienie, jeśli zmieni zdanie podczas procesu składania zamówienia.



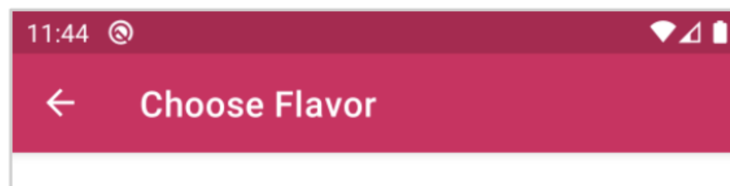
Następnie rozszerzysz aplikację, tak aby stuknięcie **Wyślij zamówienie do innej aplikacji** współdzieliło zamówienie z inną aplikacją. Następnie zamówienie można wysłać np. do sklepu z babeczkami poprzez e-mail.



Zanurzmy się i uzupełnij aplikację **Cupcake** !

3. Zaimplementuj zachowanie przycisku w górę

W aplikacji **Cupcake** pasek aplikacji pokazuje strzałkę umożliwiającą powrót do poprzedniego ekranu. Jest to znane jako przycisk w **górę** i nauczyłeś się go w poprzednich ćwiczeniach z programowania. Przycisk W **górę** obecnie nic nie robi, więc najpierw napraw ten błąd nawigacji w aplikacji.



1. W `MainActivity`, powinieneś już mieć kod, aby skonfigurować pasek aplikacji (znany również jako pasek akcji) z kontrolerem nawigacji. Utwórz `navController`zmienną klasy, aby móc jej użyć w innej metodzie.

```
class MainActivity : AppCompatActivity(R.layout.activity_main) {

    private lateinit var navController: NavController
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val navHostFragment = supportFragmentManager
                .findFragmentById(R.id.nav_host_fragment) as NavHostFragment
        navController = navHostFragment.navController

        setupActionBarWithNavController(navController)
    }
}
```

2. W tej samej klasie dodaj kod, aby zastąpić `onSupportNavigateUp()`funkcję. Ten kod poprosi `navController`o obsługę nawigacji w aplikacji. W przeciwnym razie wróć do implementacji superklasy (w `AppCompatActivity`) obsługi przycisku **W górę** .

```
override fun onSupportNavigateUp(): Boolean {
   return navController.navigateUp() || super.onSupportNavigateUp()
}
```

3. Uruchom aplikację. Przycisk W **górę** powinien teraz działać z `FlavorFragment`, `PickupFragment`i `SummaryFragment`. W miarę przechodzenia do poprzednich kroków w przepływie zamówienia fragmenty powinny pokazywać właściwy smak i datę odbioru z modelu widoku.

4. Dowiedz się o zadaniach i stosie wstecznym

Teraz wprowadzisz przycisk **Anuluj** w przepływie zamówienia swojej aplikacji. Anulowanie zamówienia w dowolnym momencie procesu zamówienia odsyła użytkownika z powrotem do `StartFragment`. Aby poradzić sobie z tym zachowaniem, dowiesz się o zadaniach i stosie wstecznym w systemie Android.

Zadania

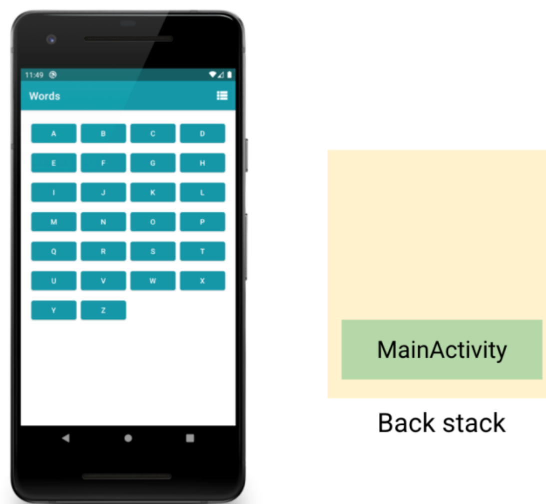
Działania w Androidzie istnieją w ramach zadań. Gdy po raz pierwszy otworzysz aplikację z ikony programu uruchamiającego, system Android utworzy nowe zadanie z Twoją główną aktywnością. Zadanie to zbiór czynności, z którymi użytkownik wchodzi w interakcję podczas wykonywania określonej pracy (np. sprawdzanie poczty, tworzenie zamówienia na babeczki, robienie zdjęcia) .

Działania są ułożone w stos, zwany *stosem tylnym* , w którym każde nowe działanie odwiedzane przez użytkownika jest umieszczane na stosie tylnym dla zadania. Możesz myśleć o tym jako o stosie naleśników, w którym każdy nowy naleśnik jest dodawany na wierzchu stosu. Aktywność na szczycie stosu to bieżąca aktywność, z którą użytkownik wchodzi w interakcję. Działania znajdujące się pod nim na stosie zostały zepchnięte na dalszy plan i zatrzymane.

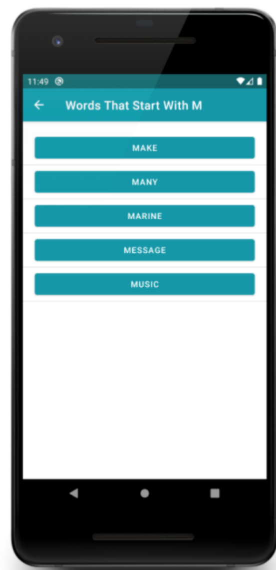
Tylny stos jest przydatny, gdy użytkownik chce nawigować wstecz. System Android może usunąć bieżące działanie ze szczytu stosu, zniszczyć je i ponownie rozpocząć działanie pod nim. Nazywa się to zdejmowaniem działania ze stosu i przenoszeniem poprzedniego działania na pierwszy plan, aby użytkownik mógł z nim wchodzić w interakcję. Jeśli użytkownik chce cofać się wiele razy, Android będzie nadal usuwał działania ze szczytu stosu, dopóki nie zbliżysz się do dna stosu. Gdy w backstack nie ma już żadnych działań, użytkownik powraca do ekranu uruchamiania urządzenia (lub do aplikacji, która uruchomiła tę).

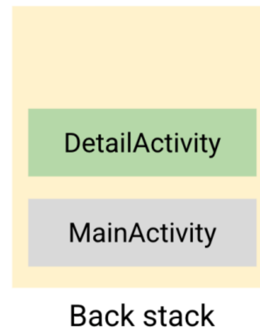
Przyjrzyjmy się wersji aplikacji **Words**`MainActivity` zaimplementowanej za pomocą 2 czynności: i `DetailActivity`.

Po pierwszym uruchomieniu aplikacji `MainActivity`otwiera się i jest dodawany do tylnego stosu zadania.

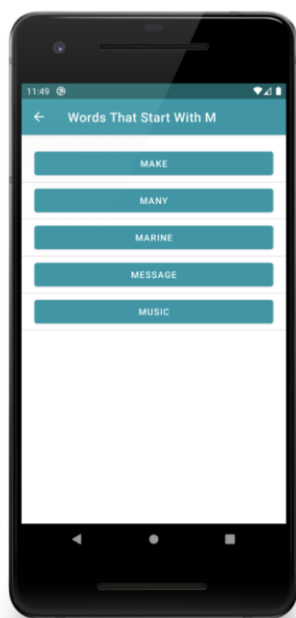


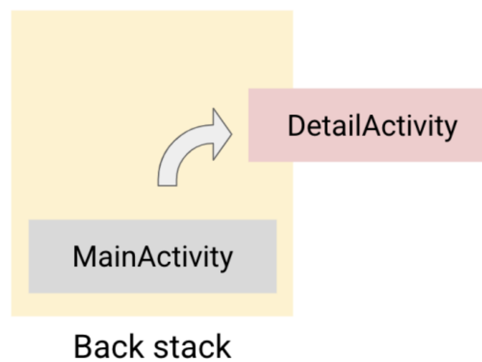
Kiedy klikniesz na literę, zostanie `DetailActivity`ona wystrzelona i wepchnięta na backstack. Oznacza to, że `DetailActivity`został utworzony, uruchomiony i wznowiony, aby użytkownik mógł z nim wchodzić w interakcje. Jest `MainActivity`umieszczony w tle i pokazany na schemacie z szarym kolorem tła.



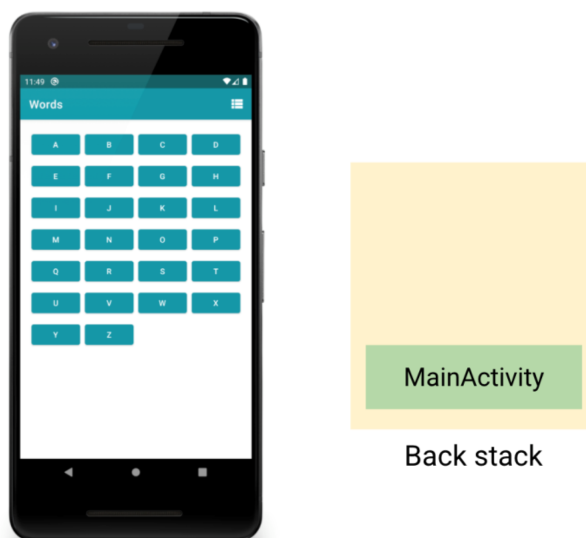


Jeśli naciśniesz przycisk **Wstecz** , instancja DetailActivityzniknie z tylnego stosu, a DetailActivityinstancja zostanie zniszczona i zakończona.



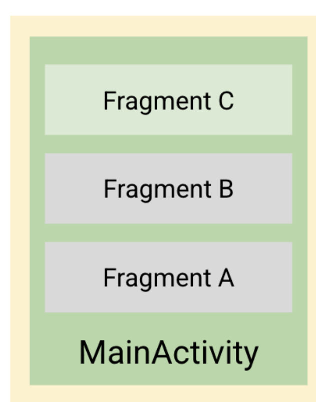


Następnie kolejny element na wierzchu tylnego stosu (MainActivity) jest wysuwany na pierwszy plan.



Uwaga: po otwarciu aplikacji, jeśli dotkniesz **Home** na urządzeniu, całe zadanie aplikacji zostanie umieszczone w tle. Jeśli ponownie dotkniesz ikony programu uruchamiającego aplikacji, system Android sprawdzi, czy istnieje zadanie dla Twojej aplikacji i przeniesie je na pierwszy plan (z nienaruszonym stosem tylnym). Jeśli nie istnieje żadne istniejące zadanie, system Android utworzy dla Ciebie nowe zadanie i uruchomi główne działanie, odkładając je na tylny stos.

W ten sam sposób, w jaki tylny stos może śledzić działania, które zostały otwarte przez użytkownika, tylny stos może również śledzić fragmenty miejsc docelowych, które odwiedził użytkownik za pomocą komponentu Jetpack Navigation.



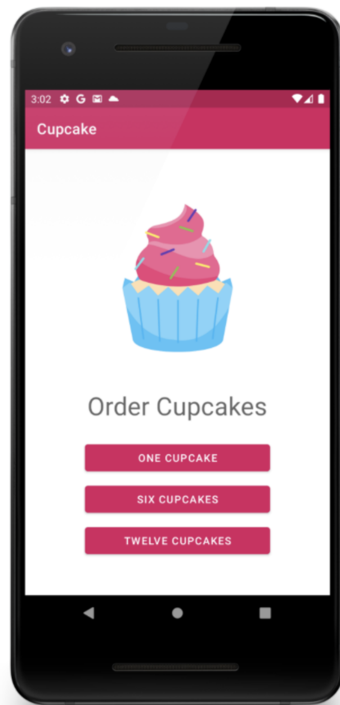
Biblioteka Nawigacyjna umożliwia usuwanie miejsca docelowego fragmentu z tylnego stosu za każdym razem, gdy użytkownik naciśnie przycisk **Wstecz** . To domyślne zachowanie jest dostępne za darmo, bez konieczności wdrażania żadnego z nich. Będziesz musiał napisać kod tylko wtedy, gdy potrzebujesz niestandardowego zachowania stosu wstecznego, które będziesz robić w aplikacji **Cupcake** .

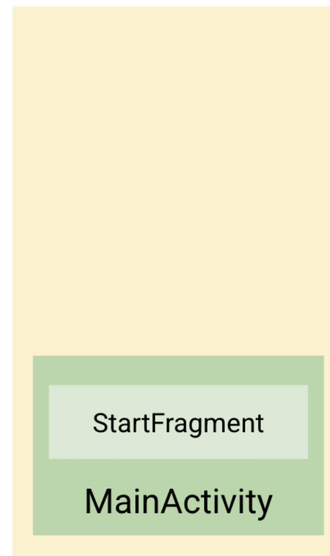
Domyślne zachowanie aplikacji Cupcake

Przyjrzyjmy się, jak działa tylny stos w aplikacji **Cupcake** . W aplikacji jest tylko jedno działanie, ale istnieje wiele miejsc docelowych fragmentów, przez które użytkownik nawiguje. Dlatego

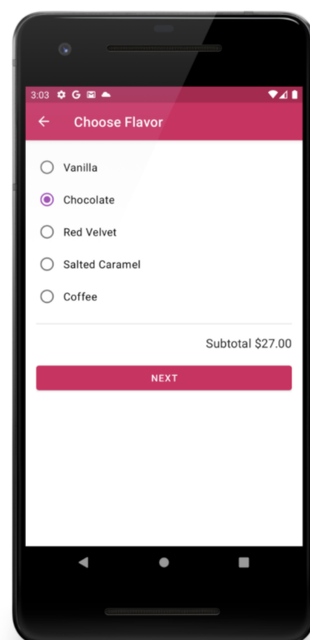
pożądane jest , aby przycisk **Wstecz** powracał do poprzedniego miejsca docelowego fragmentu za każdym dotknięciem.

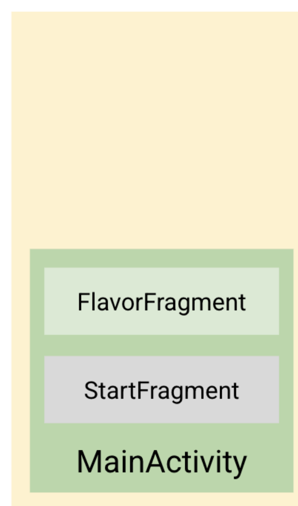
Gdy po raz pierwszy otworzysz aplikację, StartFragmentpojawi się miejsce docelowe. To miejsce docelowe zostaje wepchnięte na szczyt stosu.



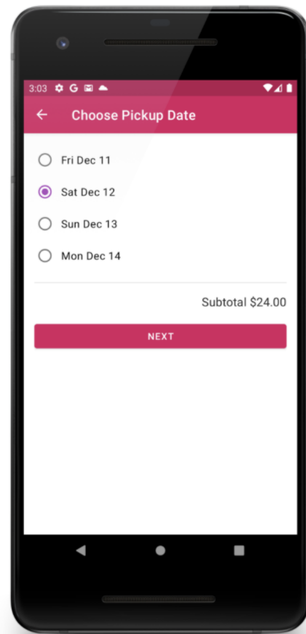


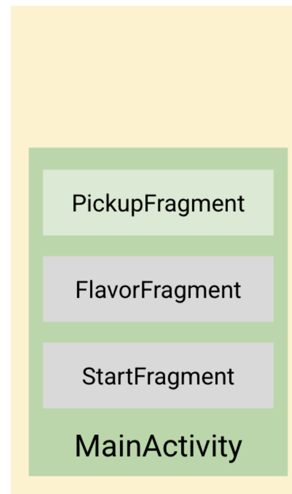
Po wybraniu liczby babeczek do zamówienia przechodzisz do ikony FlavorFragment, która jest umieszczana na tylnym stosie.



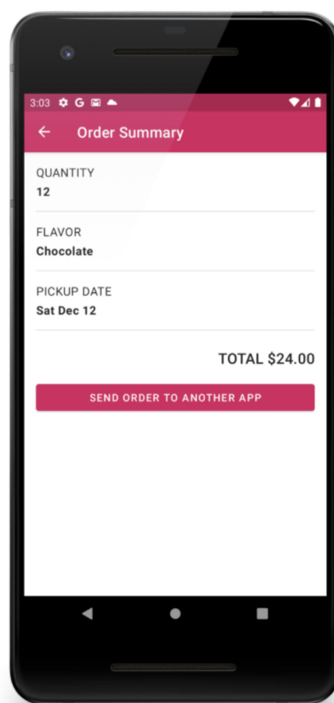


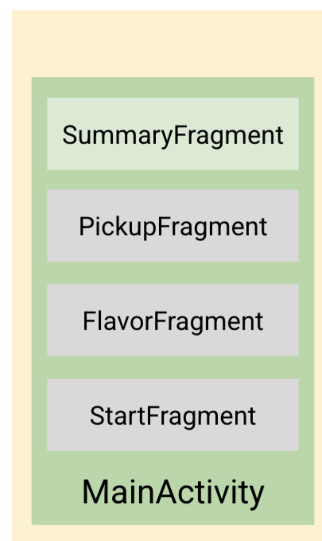
Gdy wybierzesz smak i klikniesz **Dalej** , przejdziesz do PickupFragment, który zostanie odsunięty na tylny stos.



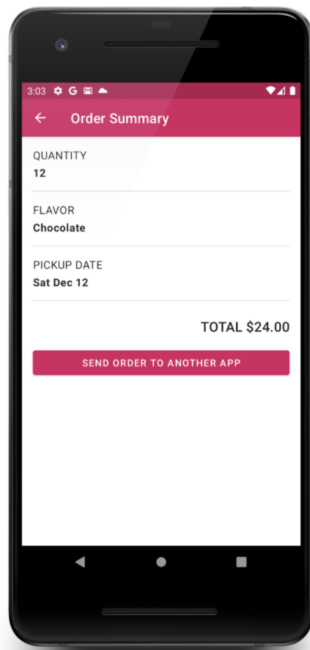


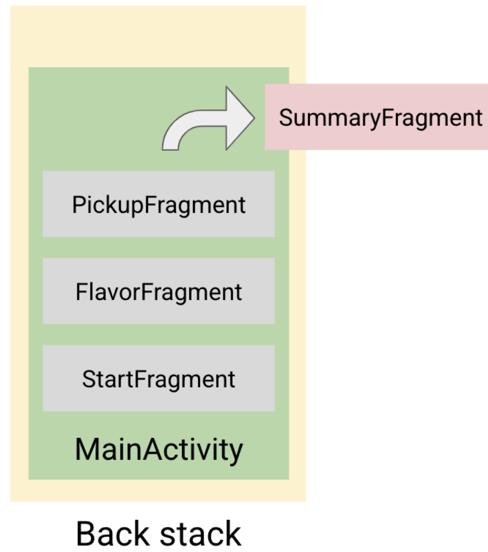
I na koniec, po wybraniu daty odbioru i dotknięciu **Dalej** , przejdziesz do `SummaryFragment`, który zostanie dodany na górze tylnego stosu.



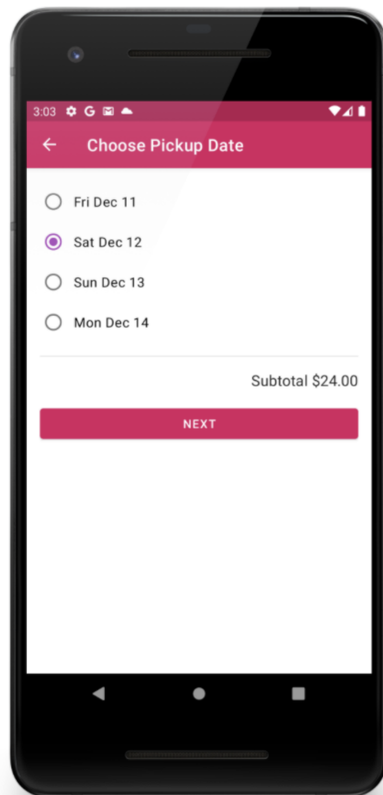


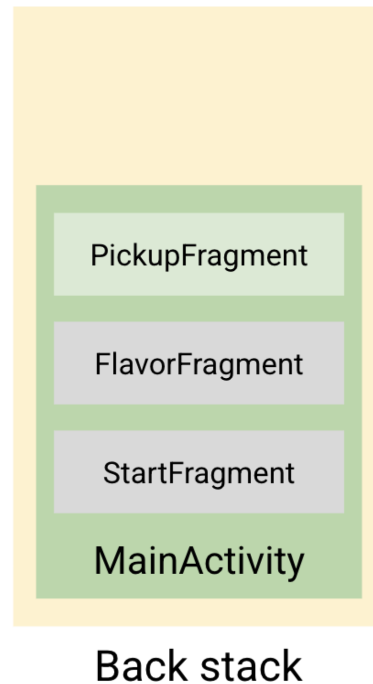
Załóżmy `SummaryFragment`, że klikasz przycisk **Wstecz** lub **W górę** . Zostają `SummaryFragment`zrzucone ze stosu i zniszczone.





Znajduje się PickupFragmentteraz na szczycie tylnego stosu i jest pokazywany użytkownikowi.

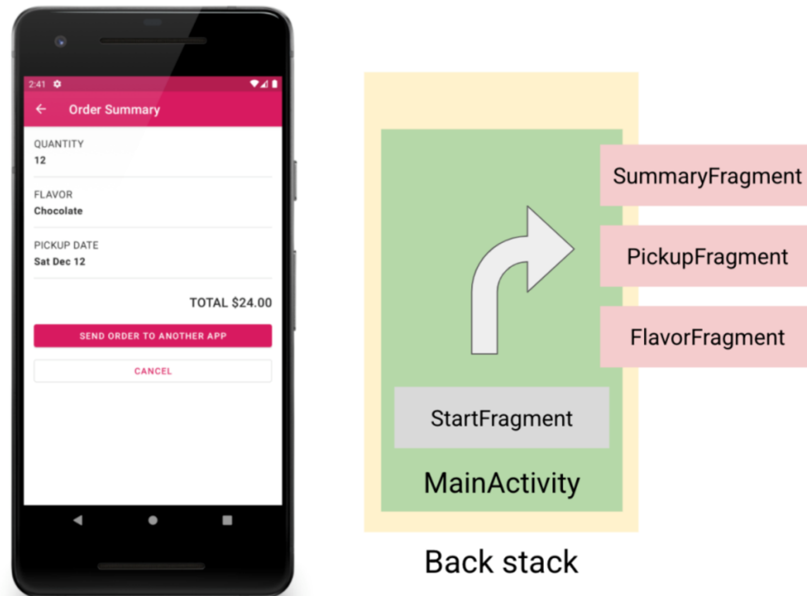




Ponownie dotknij przycisku **Wstecz** lub **W górę** . Zdejmuje się PickupFragmentze stosu, a następnie FlavorFragmentpokazuje.

Ponownie dotknij przycisku **Wstecz** lub **W górę** . Zdejmuje się FlavorFragmentze stosu, a następnie StartFragmentpokazuje.

Podczas nawigowania wstecz do wcześniejszych kroków w przepływie zamówienia, w danym momencie wyskakuje tylko jedno miejsce docelowe. Ale w następnym zadaniu dodasz do aplikacji funkcję anulowania zamówienia. Może to wymagać wyrzucenia wielu miejsc docelowych na tylnym stosie jednocześnie, aby powrócić do użytkownika i StartFragmentrozpocząć nowe zamówienie.



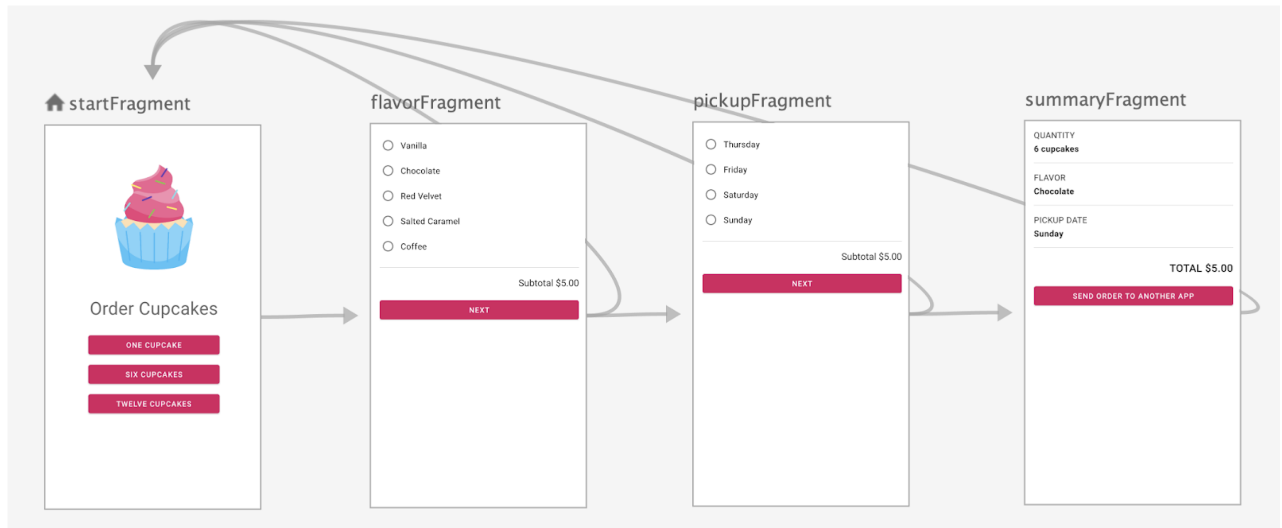
Zmodyfikuj tylny stos w aplikacji Cupcake

Zmodyfikuj klasy FlavorFragment, PickupFragment, SummaryFragmenti pliki układu, aby udostępnić użytkownikowi przycisk Anuluj zamówienie.

Dodaj akcję nawigacji

Najpierw dodaj działania nawigacji do wykresu nawigacji w swojej aplikacji, aby użytkownik mógł wrócić do StartFragmentkolejnych miejsc docelowych.

1. Otwórz **Edytor nawigacji** , przechodząc do pliku **res > nawigacji > nav_graph.xml** i wybierając widok **Projekt** .
2. Obecnie istnieje akcja od startFragmentdo flavorFragment, akcja od flavorFragmentdo pickupFragmenti akcja od pickupFragmentdo summaryFragment.
3. Kliknij i przeciągnij, aby utworzyć nową akcję nawigacji z summaryFragmentdo startFragment. Możesz zobaczyć te instrukcje , jeśli chcesz odświeżyć sposób łączenia miejsc docelowych na wykresie nawigacji.
4. Z pickupFragment, kliknij i przeciągnij, aby utworzyć nową akcję do startFragment.
5. Z flavorFragment, kliknij i przeciągnij, aby utworzyć nową akcję do startFragment.
6. Po zakończeniu wykres nawigacji powinien wyglądać następująco.



Dzięki tym zmianom użytkownik może przejść od jednego z późniejszych fragmentów przepływu zamówienia z powrotem do początku przepływu zamówienia. Teraz potrzebujesz kodu, który faktycznie nawiguje za pomocą tych działań. Właściwym miejscem jest dotknięcie przycisku **Anuluj** .

Dodaj przycisk Anuluj do układu

Najpierw dodaj przycisk **Anuluj** do plików układu dla wszystkich fragmentów z wyjątkiem `StartFragment`. Nie ma potrzeby anulowania zamówienia, jeśli jesteś już na pierwszym ekranie przepływu zamówienia.

1. Otwórz `fragment_flavor.xml`plik układu.
2. Użyj widoku **Split** , aby bezpośrednio edytować plik XML i wyświetlać podgląd obok siebie.
3. Dodaj przycisk **Anuluj** między widokiem tekstu sumy częściowej a przyciskiem **Dalej** . Przypisz mu identyfikator zasobu `@+id/cancel_button`z tekstem, który ma być wyświetlany jako `@string/cancel`.

   Przycisk powinien być umieszczony poziomo obok przycisku **Dalej** , tak aby był wyświetlany jako rząd przycisków. W przypadku wiązania pionowego powiąż górną część przycisku **Anuluj** z górną częścią przycisku **Dalej** . W przypadku więzów poziomych powiąż początek przycisku **Anuluj** z kontenerem nadrzędnym, a jego koniec z początkiem przycisku **Dalej** .

   Nadaj również przyciskowi **Anuluj** wysokość `wrap_content`i szerokość `0dp`, aby mógł równo podzielić szerokość ekranu z drugim przyciskiem. Pamiętaj, że przycisk nie będzie widoczny w okienku **podglądu** do następnego kroku.

```
...

<TextView
    android:id="@+id/subtotal" ... />

<Button
    android:id="@+id/cancel_button"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="@string/cancel"
    app:layout_constraintEnd_toStartOf="@id/next_button"
```

```
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="@id/next_button" />

<Button
    android:id="@+id/next_button" ... />

...
```

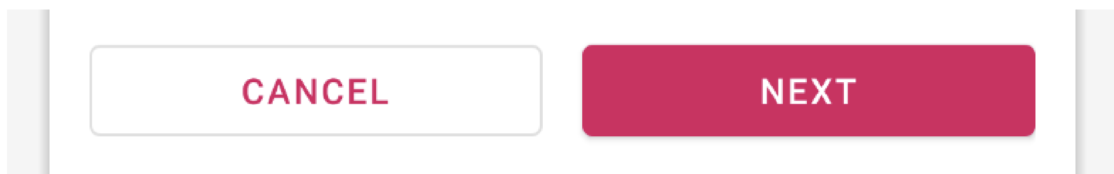
4. W fragment_flavor.xmlprogramie musisz również zmienić ograniczenie początku przycisku **Dalej** z app:layout_constraintStart_toStartOf="parent„ na app:layout_constraintStart_toEndOf="@id/cancel_button". Dodaj także margines końcowy na przycisku **Anuluj** , aby między dwoma przyciskami było trochę odstępu. Teraz przycisk **Anuluj** powinien pojawić się w okienku **Podgląd** w Studio Androida.

```
...

<Button
    android:id="@+id/cancel_button"
    android:layout_marginEnd="@dimen/side_margin" ... />

<Button
    android:id="@+id/next_button"
    app:layout_constraintStart_toEndOf="@id/cancel_button"... />

...
```

5. Jeśli chodzi o styl wizualny, zastosuj styl [przycisku Obrys materiału]() (z atrybutem style="?attr/materialButtonOutlinedStyle"), aby przycisk **Anuluj** nie był widoczny w porównaniu z przyciskiem **Dalej** , który jest podstawową akcją, na której użytkownik ma się skupić.

```
<Button
    android:id="@+id/cancel_button"
    style="?attr/materialButtonOutlinedStyle" ... />
```

Przycisk i pozycjonowanie wyglądają teraz świetnie!



6. W ten sam sposób dodaj przycisk **Anuluj** do fragment_pickup.xmlpliku układu.

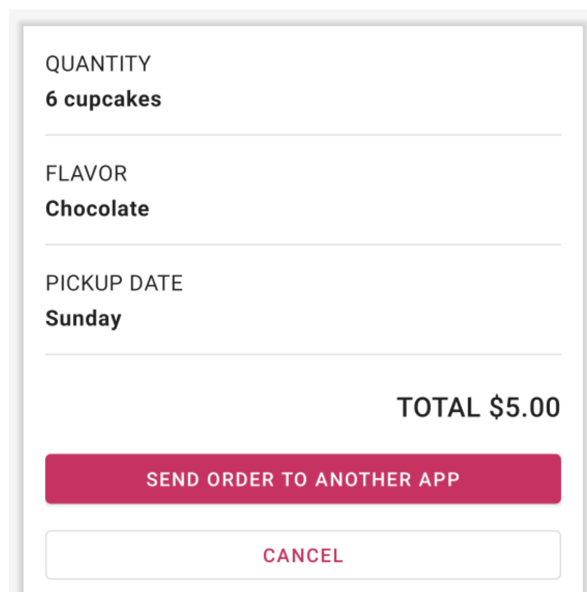
```
...

<TextView
    android:id="@+id/subtotal" ... />
```

```
<Button
   android:id="@+id/cancel_button"
   style="?attr/materialButtonOutlinedStyle"
   android:layout_width="0dp"
   android:layout_height="wrap_content"
   android:layout_marginEnd="@dimen/side_margin"
   android:text="@string/cancel"
   app:layout_constraintEnd_toStartOf="@id/next_button"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintTop_toTopOf="@id/next_button" />

<Button
   android:id="@+id/next_button" ... />

...
```

7. Zaktualizuj również ograniczenie początkowe na przycisku **Dalej** . Następnie w podglądzie pojawi się przycisk **Anuluj .**

```
<Button
   android:id="@+id/next_button"
   app:layout_constraintStart_toEndOf="@id/cancel_button" ... />
```

8. Zastosuj podobną zmianę do `fragment_summary.xml`pliku, chociaż układ tego fragmentu jest nieco inny. Dodasz przycisk **Anuluj** poniżej przycisku **Wyślij** w pionie nadrzędnym `LinearLayout`z pewnym marginesem pomiędzy.




```
...

   <Button
      android:id="@+id/send_button" ... />
```

```
<Button
    android:id="@+id/cancel_button"
    style="?attr/materialButtonOutlinedStyle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="@dimen/margin_between_elements"
    android:text="@string/cancel" />

</LinearLayout>
```

9. Uruchom i przetestuj aplikację. Powinieneś teraz zobaczyć przycisk **Anuluj** pojawiający się w układach dla `FlavorFragment`, `PickupFragment`i `SummaryFragment`. Jednak dotknięcie przycisku jeszcze nic nie robi. Skonfiguruj detektory kliknięć dla tych przycisków w następnym kroku.

Dodaj przycisk Anuluj kliknij słuchacz

W każdej klasie fragmentów (z wyjątkiem `StartFragment`) dodaj metodę pomocniczą, która obsługuje po kliknięciu przycisku **Anuluj .**

1. Dodaj tę `cancelOrder()`metodę do `FlavorFragment`. Po wyświetleniu opcji smaku, jeśli użytkownik zdecyduje się anulować zamówienie, wyczyść model widoku, wywołując `sharedViewModel.resetOrder().`Następnie przejdź z powrotem do `StartFragment`akcji nawigacji z identyfikatorem`R.id.action_flavorFragment_to_startFragment.`

```
fun cancelOrder() {
    sharedViewModel.resetOrder()
    findNavController().navigate(R.id.action_flavorFragment_to_startFragment)
}
```

Jeśli zobaczysz błąd związany z identyfikatorem zasobu akcji, być może trzeba będzie wrócić do `nav_graph.xml`pliku, aby sprawdzić, czy akcje nawigacji mają taką samą nazwę (`action_flavorFragment_to_startFragment`).

2. Użyj powiązania odbiornika, aby skonfigurować odbiornik kliknięcia na przycisku **Anuluj** w `fragment_flavor.xml`układzie. Kliknięcie tego przycisku wywoła `cancelOrder()`metodę, którą właśnie utworzyłeś w `FragmentFlavor`klasie.

```
<Button
    android:id="@+id/cancel_button"
    android:onClick="@{() -> flavorFragment.cancelOrder()}" ... />
```

3. Powtórz ten sam proces dla `PickupFragment`. Dodaj `cancelOrder()`metodę do klasy fragmentu, która resetuje kolejność i przechodzi od `PickupFragment`do `StartFragment`.

```
fun cancelOrder() {
    sharedViewModel.resetOrder()
    findNavController().navigate(R.id.action_pickupFragment_to_startFragment)
}
```

4. W fragment_pickup.xmlprogramie ustaw detektor kliknięć na przycisku **Anuluj** , aby wywoływał cancelOrder()metodę po kliknięciu.

```
<Button
    android:id="@+id/cancel_button"
    android:onClick="@{() -> pickupFragment.cancelOrder()}" ... />
```

5. Dodaj podobny kod dla przycisku **Anuluj** w SummaryFragment, przenosząc użytkownika z powrotem do StartFragment. Może być konieczne zaimportowanie androidx.navigation.fragment.findNavController, jeśli nie zostanie ono automatycznie zaimportowane dla Ciebie.

```
fun cancelOrder() {
    sharedViewModel.resetOrder()
    findNavController().navigate(R.id.action_summaryFragment_to_startFragment)
}
```

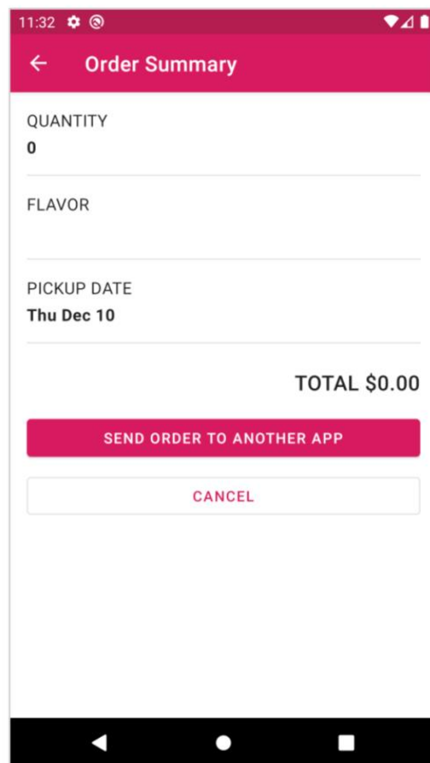
6. W fragment_summary.xmlprogramie wywołaj metodę po SummaryFragmentkliknięciu przycisku **Anuluj .**cancelOrder()

```
<Button
    android:id="@+id/cancel_button"
    android:onClick="@{() -> summaryFragment.cancelOrder()}" ... />
```

7. Uruchom i przetestuj aplikację, aby zweryfikować logikę dodaną właśnie do każdego fragmentu. Podczas tworzenia zamówienia babeczek stuknięcie przycisku **Anuluj** na FlavorFragment, PickupFragmentlub SummaryFragmentpowoduje powrót do StartFragment. Kontynuując tworzenie nowego zamówienia, powinieneś zauważyć, że informacje z poprzedniego zamówienia zostały usunięte.

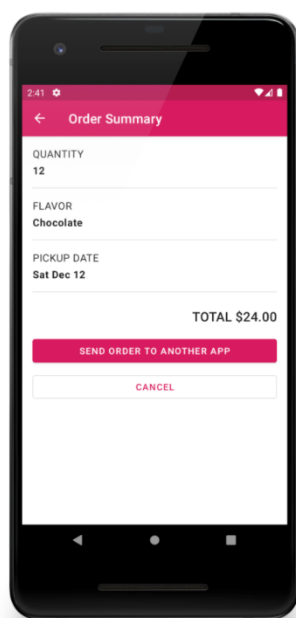
   Wygląda na to, że to działa, ale w rzeczywistości jest błąd związany z nawigacją wstecz po powrocie do StartFragment. Wykonaj kilka następnych kroków, aby odtworzyć błąd.

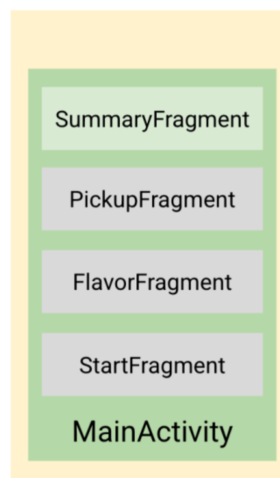
8. Przejdź przez kolejność zamówień, aby utworzyć nowe zamówienie na babeczki, aż dotrzesz do ekranu podsumowania. Na przykład możesz zamówić 12 babeczek o smaku czekoladowym i wybrać przyszłą datę odbioru.
9. Następnie stuknij **Anuluj** . Powinieneś wrócić do StartFragment.
10. Wygląda to poprawnie, ale jeśli naciśniesz systemowy przycisk **Wstecz** , wrócisz do ekranu podsumowania zamówienia z podsumowaniem zamówienia dla 0 babeczek i bez smaku. To jest nieprawidłowe i nie powinno być pokazywane użytkownikowi.



Użytkownik prawdopodobnie nie chce wracać do przepływu zamówień. Ponadto wszystkie dane zamówień w modelu widoku zostały wyczyszczone, więc te informacje nie są przydatne. Zamiast tego stuknięcie przycisku **Wstecz** z StartFragmentpowinno opuścić aplikację **Cupcake** .

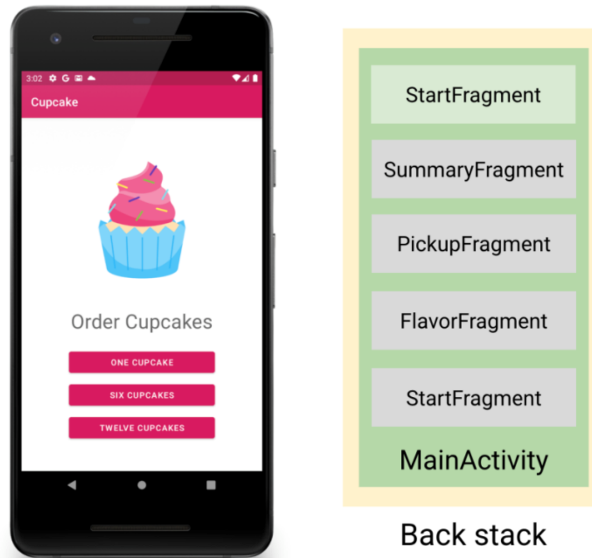
Przyjrzyjmy się, jak obecnie wygląda tylny stos i jak naprawić błąd. Kiedy tworzysz zamówienie na ekranie podsumowania zamówienia, każde miejsce docelowe jest umieszczane na tylnym stosie.





Back stack

Od SummaryFragmentdnia anulowałeś zamówienie. Podczas nawigowania za pomocą akcji z SummaryFragmentdo system StartFragmentAndroid dodał kolejną instancję StartFragmentjako nowe miejsce docelowe na tylnym stosie.



Dlatego po dotknięciu przycisku **Wstecz** z StartFragment, aplikacja wyświetliła SummaryFragmentponownie (z pustymi informacjami o zamówieniu).

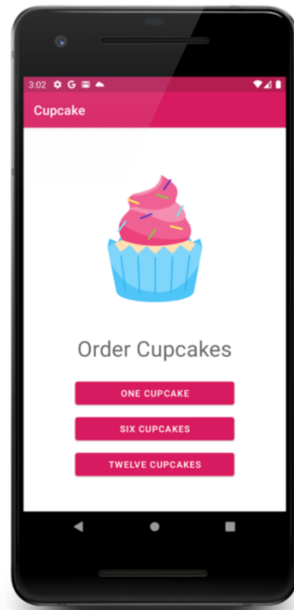
Aby naprawić ten błąd nawigacji, dowiedz się, jak składnik Nawigacja umożliwia usuwanie dodatkowych miejsc docelowych z tylnego stosu podczas nawigacji za pomocą akcji.

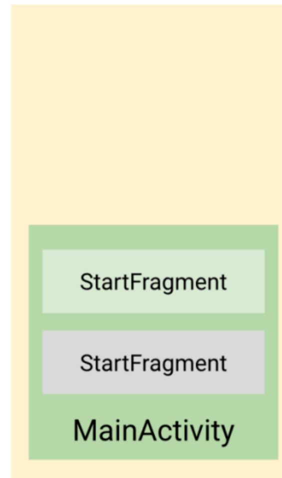
Zdejmij dodatkowe miejsca docelowe z tylnego stosu

Akcja nawigacji: atrybut popUpTo

Dołączając app:popUpToatrybut do akcji nawigacyjnej na wykresie nawigacyjnym, więcej niż jedno miejsce docelowe może zostać usunięte z tylnego stosu, aż do osiągnięcia określonego miejsca docelowego. Jeśli określisz app:popUpTo="@id/startFragment", miejsca docelowe z tylnego stosu zostaną usunięte, dopóki nie osiągniesz StartFragment, który pozostanie na stosie.

Gdy dodasz tę zmianę do kodu i uruchomisz aplikację, zauważysz, że po anulowaniu zamówienia wrócisz do StartFragment. Ale tym razem, po dotknięciu przycisku **Wstecz** z StartFragment, zobaczysz StartFragmentponownie (zamiast wychodzić z aplikacji). To również nie jest pożądane zachowanie. Jak wspomniano wcześniej, odkąd przechodzisz do StartFragment, system Android faktycznie dodaje StartFragmentnowe miejsce docelowe na tylnym stosie, więc teraz masz 2 wystąpienia StartFragment na tylnym stosie. Dlatego musisz dwukrotnie nacisnąć przycisk **Wstecz** , aby wyjść z aplikacji.

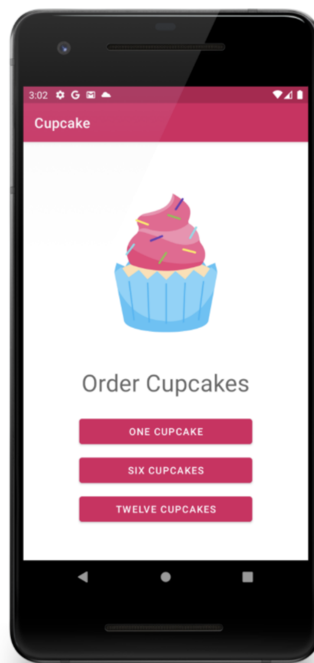


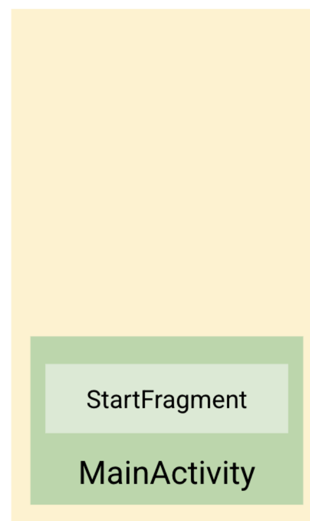


Akcja nawigacji: atrybut popUpToInclusive

Aby naprawić ten nowy błąd, poproś, aby wszystkie miejsca docelowe zostały usunięte z tylnego stosu, aż do `StartFragment`. Zrób to, określając`app:popUpTo="@id/startFragment"`

oraz `app:popUpToInclusive="true"`na odpowiednich działaniach nawigacyjnych. W ten sposób będziesz mieć tylko jedną nową instancję `StartFragment`na tylnym stosie. Następnie jednokrotne dotknięcie przycisku **Wstecz**`StartFragment` z spowoduje wyjście z aplikacji. Dokonajmy teraz tej zmiany.

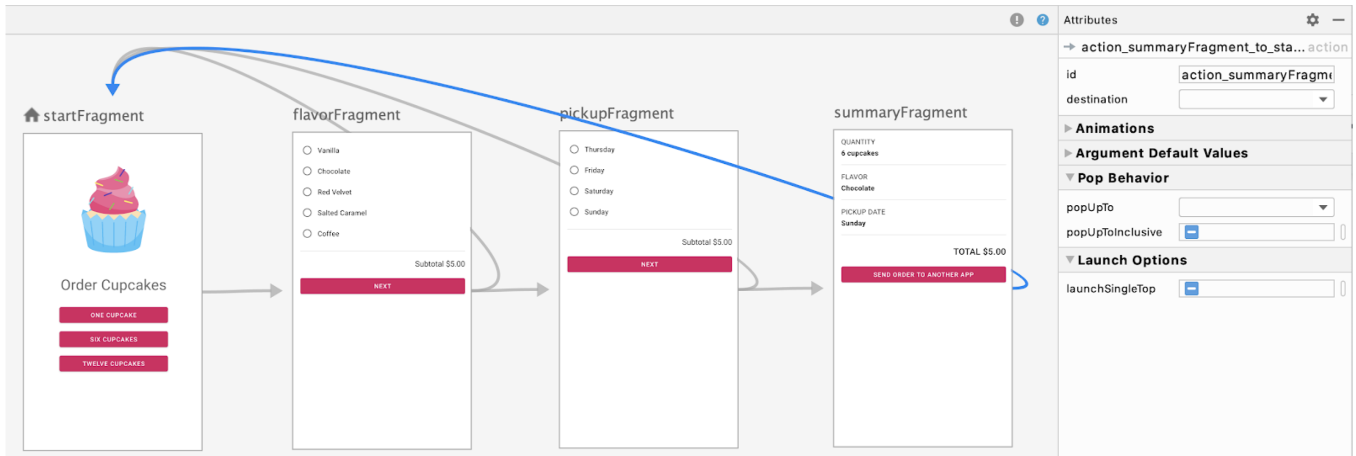




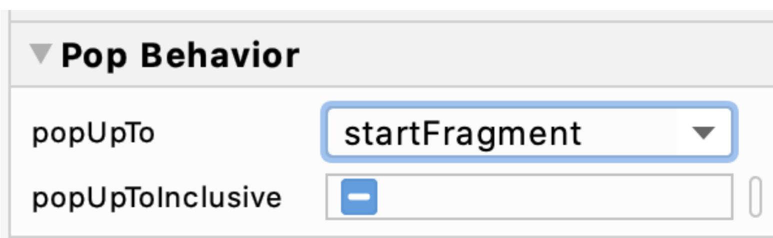
Zmodyfikuj działania nawigacyjne

1. Przejdź do **edytora nawigacji** , otwierając plik **res>nawigacja>nav_graph.xml** .

2. Wybierz akcję, która przechodzi od `summaryFragment`do `startFragment`, tak aby była podświetlona na niebiesko.
3. Rozwiń **atrybuty** po prawej stronie (jeśli nie są jeszcze otwarte). Poszukaj **Pop Behavior** na liście atrybutów, które możesz zmodyfikować.



4. Z rozwijanych opcji ustaw **popUpTo** na `startFragment`. Oznacza to, że wszystkie miejsca docelowe w tylnym stosie zostaną usunięte (zaczynając od góry stosu i przesuwając się w dół), aż do `startFragment`.



5. Następnie kliknij pole wyboru **popUpToInclusive** , aż pojawi się znacznik wyboru i etykieta **true** . Oznacza to, że chcesz usunąć miejsca docelowe do instancji `startFragment`, która już znajduje się na tylnym stosie włącznie. Wtedy nie będziesz mieć dwóch wystąpień `startFragment`na tylnym stosie.

6. Powtórz te zmiany dla akcji łączącej `pickupFragment`się z `startFragment`.

7. Powtórz dla akcji łączącej `flavorFragment`się z `startFragment`.

8. Gdy skończysz, potwierdź, że wprowadziłeś prawidłowe zmiany w swojej aplikacji, patrząc na widok **Kod** w pliku wykresu nawigacji.

```
<navigation
    android:id="@+id/nav_graph" ...>
    <fragment
        android:id="@+id/startFragment" ...>
        ...
    </fragment>
    <fragment
        android:id="@+id/flavorFragment" ...>
        ...
        <action
            android:id="@+id/action_flavorFragment_to_startFragment"
            app:destination="@id/startFragment"
            app:popUpTo="@id/startFragment"
            app:popUpToInclusive="true" />
    </fragment>
    <fragment
        android:id="@+id/pickupFragment" ...>
        ...
        <action
            android:id="@+id/action_pickupFragment_to_startFragment"
            app:destination="@id/startFragment"
            app:popUpTo="@id/startFragment"
            app:popUpToInclusive="true" />
    </fragment>
    <fragment
        android:id="@+id/summaryFragment" ...>
        <action
            android:id="@+id/action_summaryFragment_to_startFragment"
            app:destination="@id/startFragment"
            app:popUpTo="@id/startFragment"
            app:popUpToInclusive="true" />
    </fragment>
</navigation>
```

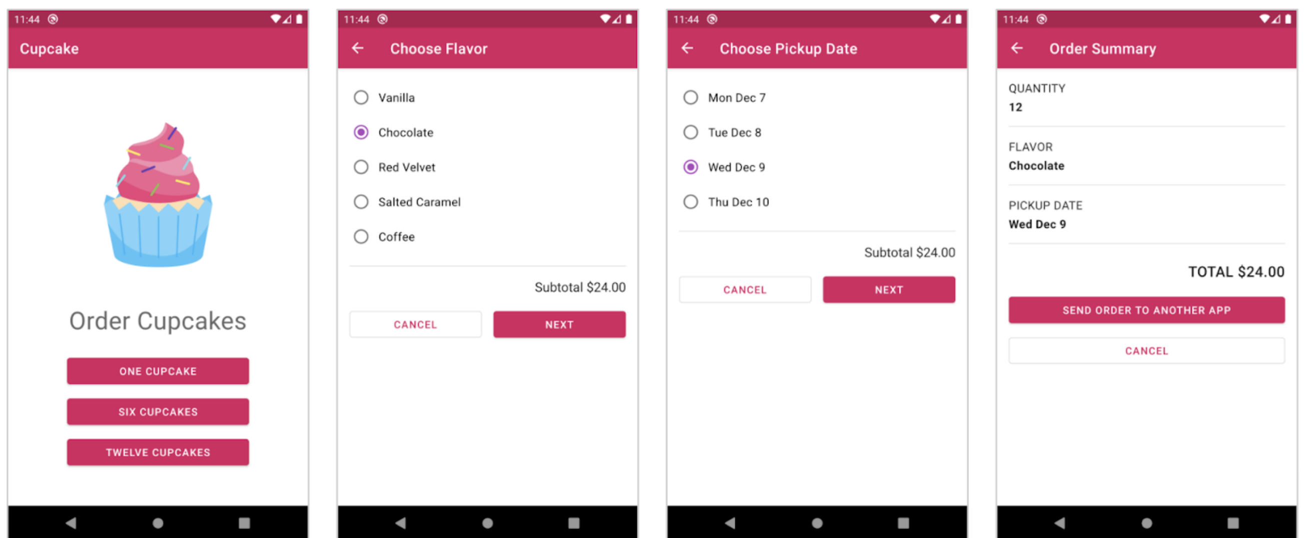
Zauważ, że dla każdej z 3 akcji (`action_flavorFragment_to_startFragment`, `action_pickupFragment_to_startFragment`i `action_summaryFragment_to_startFragment`) powinny być dodane nowe atrybuty `app:popUpTo="@id/startFragment"`i `app:popUpToInclusive="true"`.

9. Teraz uruchom aplikację. Przejrzyj kolejność zamówień i kliknij **Anuluj** . Po powrocie do `StartFragment`, dotknij przycisku **Wstecz** (tylko raz!) i opuść aplikację.

   Podsumowując, co się dzieje, gdy anulowałeś zamówienie i wróciłeś do pierwszego ekranu aplikacji, wszystkie miejsca docelowe fragmentów w tylnym stosie zostały usunięte ze stosu, w tym pierwsze wystąpienie `StartFragment`. Po zakończeniu akcji nawigacyjnej `StartFragment`został dodany jako nowy cel podróży na tylnym stosie. Tapping **Back** stamtąd wyskakuje `StartFragment`ze

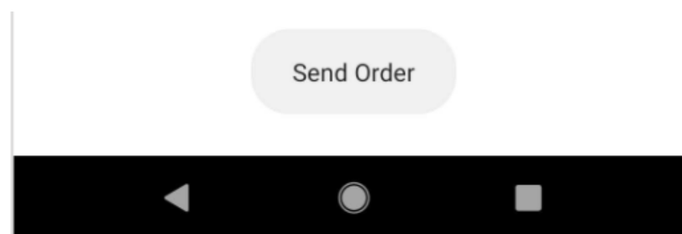
stosu, nie pozostawiając więcej fragmentów w tylnym stosie. Dlatego Android kończy działanie, a użytkownik opuszcza aplikację.

Oto jak powinna wyglądać aplikacja:

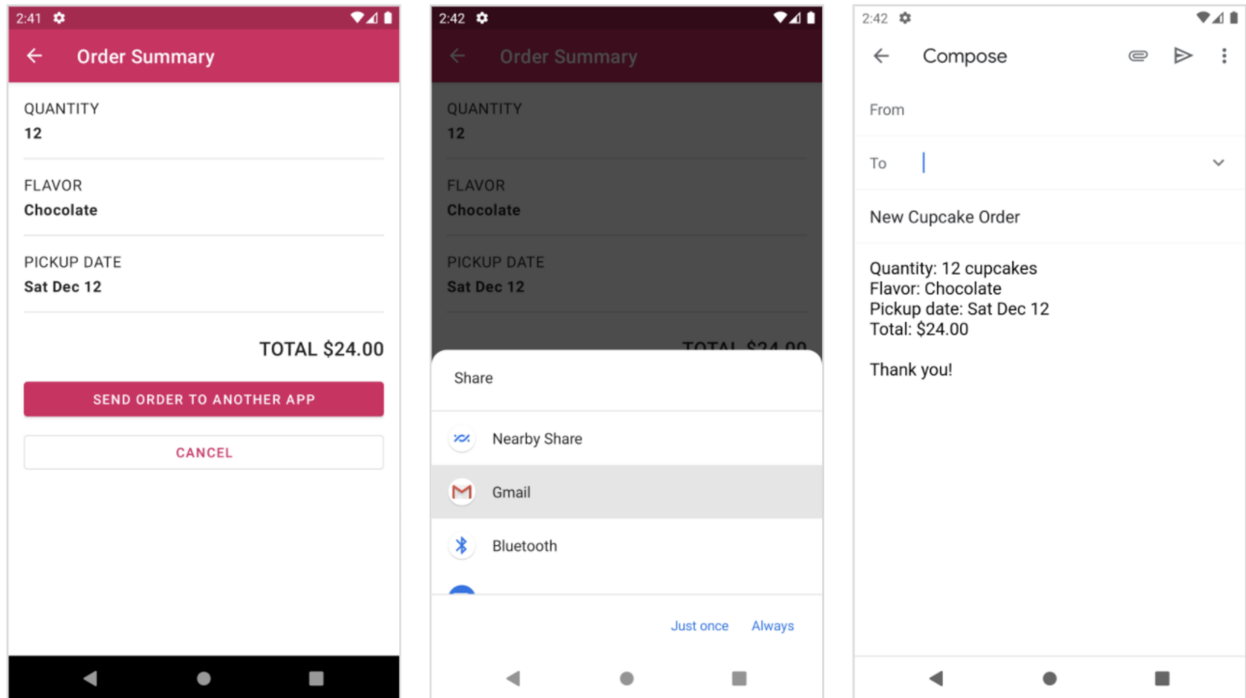


5. Wyślij zamówienie

Jak dotąd aplikacja wygląda fantastycznie! Brakuje jednak jednej części. Po dotknięciu przycisku wyślij zamówienie na SummaryFragment, pojawi Toastsię komunikat.



Byłoby bardziej przydatnym doświadczeniem, gdyby zamówienie mogło zostać wysłane z aplikacji. Skorzystaj z tego, czego nauczyłeś się we wcześniejszych ćwiczeniach z programowania na temat używania niejawnej intencji do udostępniania informacji z aplikacji innej aplikacji. W ten sposób użytkownik może udostępnić informacje o zamówieniu babeczek aplikacji e-mail na urządzeniu, umożliwiając wysłanie zamówienia do sklepu z babeczkami.



Aby wdrożyć tę funkcję, spójrz na strukturę tematu i treści wiadomości e-mail na powyższym zrzucie ekranu.

Będziesz używać tych ciągów, które są już w Twoim `strings.xml`pliku.

```
<string name="new_cupcake_order">New Cupcake Order</string>
<string name="order_details">Quantity: %1$s cupcakes \n Flavor: %2$s \nPickup date: %3$s \n Total:
%4$s \n\n Thank you!</string>
```

`order_details`to zasób tekstowy z 4 różnymi argumentami formatu, które są symbolami zastępczymi dla rzeczywistej ilości babeczek, pożądanego smaku, pożądanej daty odbioru i całkowitej ceny. Argumenty są ponumerowane od 1 do 4 ze składnią `%1`do `%4`. Określony jest również typ argumentu (`$s`oznacza, że oczekiwany jest tutaj ciąg).

W kodzie Kotlina będziesz mógł wywołać następujące `getString()`po `R.string.order_details`nich 4 argumenty (kolejność ma znaczenie!). Na przykład wywołanie `getString(R.string.order_details, "12", "Chocolate", "Sat Dec 12", "$24.00")`tworzy następujący ciąg, który jest dokładnie taką treścią wiadomości e-mail, jaką chcesz.

```
Quantity: 12 cupcakes
Flavor: Chocolate
Pickup date: Sat Dec 12
Total: $24.00

Thank you!
```

Uwaga: być może pamiętasz, że znasz już zasoby ciągów z argumentami formatu. Na przykład ciągi ceny częściowej i całkowitej, których używasz w aplikacji, są zadeklarowane jako

```
<string name="subtotal_price">Subtotal %s</string>
<string name="total_price">Total %s</string>
```

gdzie %sjest symbolem zastępczym dla sformatowanego ciągu ceny.

1. Zmodyfikuj SummaryFragment.ktmetodę sendOrder(). Usuń istniejącą Toastwiadomość.

```
fun sendOrder() {

}
```

2. W ramach sendOrder()metody skonstruuj tekst podsumowania zamówienia. Utwórz sformatowany order_detailsciąg, pobierając ilość zamówienia, smak, datę i cenę z modelu widoku współdzielonego.

```
val orderSummary = getString(
   R.string.order_details,
   sharedViewModel.quantity.value.toString(),
   sharedViewModel.flavor.value.toString(),
   sharedViewModel.date.value.toString(),
   sharedViewModel.price.value.toString()
)
```

3. Nadal w ramach sendOrder()metody utwórz niejawny zamiar udostępnienia zamówienia innej aplikacji. Zapoznaj się z dokumentacją , aby dowiedzieć się, jak utworzyć intencję e-mail. Określ Intent.ACTION_SENDzamierzone działanie, ustaw typ na "text/plain"i uwzględnij dodatkowe intencje w temacie wiadomości e-mail (Intent.EXTRA_SUBJECT) i treści wiadomości e-mail (Intent.EXTRA_TEXT). android.content.IntentW razie potrzeby importuj .

```
val intent = Intent(Intent.ACTION_SEND)
   .setType("text/plain")
   .putExtra(Intent.EXTRA_SUBJECT, getString(R.string.new_cupcake_order))
   .putExtra(Intent.EXTRA_TEXT, orderSummary)
```

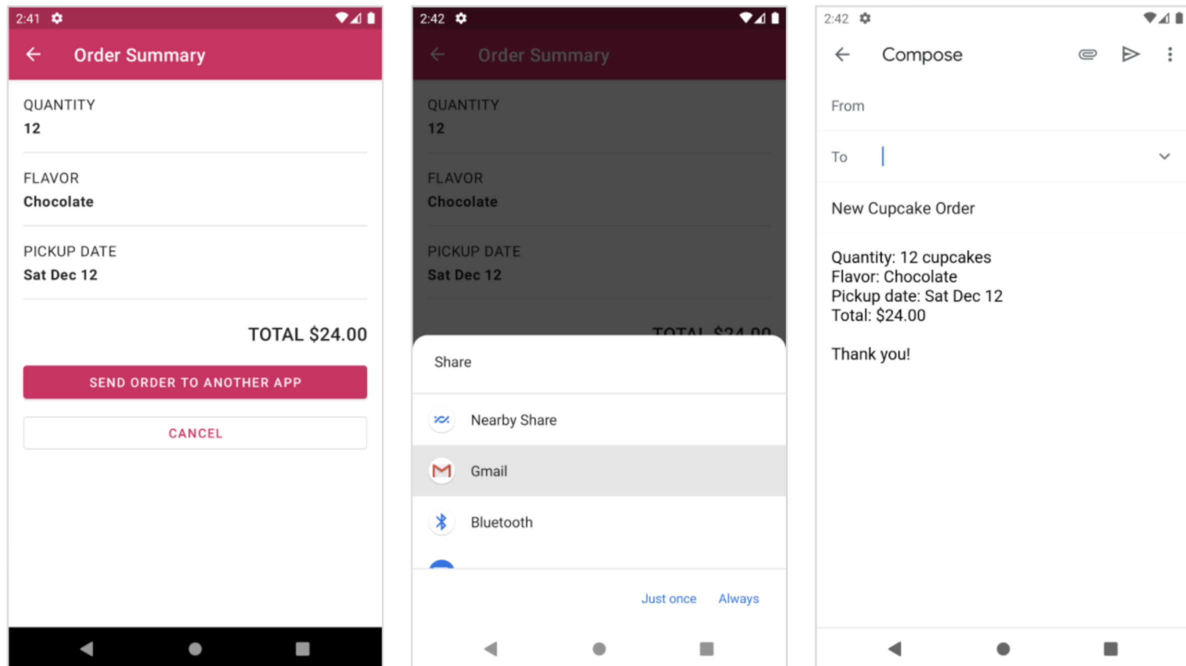
   Jako dodatkowa wskazówka, jeśli dostosujesz tę aplikację do własnego przypadku użycia, możesz wstępnie wypełnić adresata wiadomości e-mail jako adres e-mail sklepu z babeczkami. W intencji możesz określić odbiorcę wiadomości e-mail z dodatkową intencją Intent.EXTRA_EMAIL.

4. Ponieważ jest to niejawna intencja, nie musisz wiedzieć z wyprzedzeniem, który konkretny składnik lub aplikacja obsłuży tę intencję. Użytkownik zdecyduje, której aplikacji chce użyć, aby zrealizować swój zamiar. Jednak przed uruchomieniem działania z tym zamiarem sprawdź, czy istnieje aplikacja, która mogłaby to obsłużyć. To sprawdzenie zapobiegnie zawieszaniu się aplikacji **Cupcake** , jeśli nie ma aplikacji do obsługi zamiaru, dzięki czemu Twój kod będzie bezpieczniejszy.

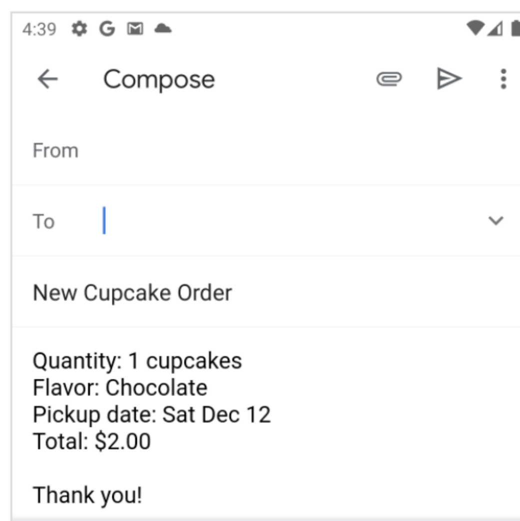
```
if (activity?.packageManager?.resolveActivity(intent, 0) != null) {
   startActivity(intent)
}
```

   Wykonaj to sprawdzenie, uzyskując dostęp do PackageManager, który zawiera informacje o tym, jakie pakiety aplikacji są zainstalowane na urządzeniu. Można PackageManageruzyskać dostęp za pośrednictwem fragmentu activity, o ile activityi packageManagernie są null. Wywołaj metodę z intencją, którą utworzyłeś PackageManager. resolveActivity()Jeśli wynik nie jest pusty, możesz bezpiecznie zadzwonić startActivity()z zamiarem.

5. Uruchom aplikację, aby przetestować kod. Utwórz zamówienie na babeczki i dotknij opcji **Wyślij zamówienie do innej aplikacji** . Gdy pojawi się okno dialogowe udostępniania, możesz wybrać aplikację Gmail, ale jeśli wolisz, możesz wybrać inną aplikację. Jeśli wybierzesz aplikację Gmail, może być konieczne skonfigurowanie konta na urządzeniu, jeśli jeszcze tego nie zrobiłeś (na przykład jeśli używasz emulatora). Jeśli w treści wiadomości e-mail nie widzisz swojego najnowszego zamówienia babeczek, być może najpierw musisz odrzucić bieżącą wersję roboczą wiadomości e-mail.



Testując różne scenariusze, możesz zauważyć błąd, jeśli masz tylko 1 ciastko. Podsumowanie zamówienia mówi **1 babeczki** , ale w języku angielskim jest to gramatycznie niepoprawne.



Zamiast tego powinno być napisane **1 ciastko** (bez liczby mnogiej). Jeśli chcesz wybrać, czy słowo cupcake lub cupcakes ma być używane na podstawie wartości ilości, możesz użyć czegoś, co nazywa się [ciągami ilościowymi](#) w Androidzie. Deklarując `plurals`zasób, możesz określić różne zasoby tekstowe, które będą używane w zależności od ilości, na przykład w liczbie pojedynczej lub mnogiej.

6. Dodaj `cupcakes`zasób liczby mnogiej w swoim `strings.xml`pliku.

```
<plurals name="cupcakes">
   <item quantity="one">%d cupcake</item>
   <item quantity="other">%d cupcakes</item>
</plurals>
```

   W przypadku liczby pojedynczej (`quantity="one"`) zostanie użyty ciąg w liczbie pojedynczej. We wszystkich innych przypadkach (`quantity="other"`) zostanie użyty ciąg w liczbie mnogiej. Zauważ, że zamiast tego `%s`oczekuje argumentu w postaci ciągu, `%d`oczekuje argumentu w postaci liczby całkowitej, który przekażesz podczas formatowania ciągu.

   W swoim kodzie Kotlin dzwoniąc:

   `getQuantityString(R.plurals.cupcakes, 1, 1)`zwraca ciąg`1 cupcake`

   `getQuantityString(R.plurals.cupcakes, 6, 6)`zwraca ciąg`6 cupcakes`

   `getQuantityString(R.plurals.cupcakes, 0, 0)`zwraca ciąg`0 cupcakes`

   Uwaga: Wywołując `getQuantityString()`, musisz podać ilość dwukrotnie, ponieważ pierwszy parametr ilość służy do wybrania poprawnego ciągu w liczbie mnogiej. Drugi parametr ilości jest używany w `%d`symbolu zastępczym rzeczywistego zasobu ciągu.

7. Przed przejściem do kodu Kotlin zaktualizuj `order_details`zasób ciągu `strings.xml`, aby liczba mnoga **babeczek** nie była już w nim zakodowana.

```
<string name="order_details">Quantity: %1$s \n Flavor: %2$s \nPickup date: %3$s \n
     Total: %4$s \n\n Thank you!</string>
```

8. W `SummaryFragment`klasie zaktualizuj `sendOrder()`metodę, aby używała nowego ciągu ilościowego. Najłatwiej byłoby najpierw ustalić wielkość z modelu widoku i zapisać ją w zmiennej. Ponieważ `quantity`w widoku model jest typu `LiveData<Int>`, możliwe jest, że `sharedViewModel.quantity.value`jest to null. Jeśli ma wartość null, użyj `0`jako wartości domyślnej dla `numberOfCupcakes`.

   Dodaj to jako pierwszy wiersz kodu w swojej `sendOrder()`metodzie.

```
val numberOfCupcakes = sharedViewModel.quantity.value ?: 0
```

   Operator elvisa (?:) oznacza, że jeśli wyrażenie po lewej nie jest puste, użyj go. W przeciwnym razie, jeśli wyrażenie po lewej stronie ma wartość null, użyj wyrażenia po prawej stronie operatora elvis (`0`w tym przypadku).

9. Następnie sformatuj `order_details`ciąg tak jak poprzednio. Zamiast przekazywać `numberOfCupcakes`bezpośrednio jako argument ilość, utwórz sformatowany łańcuch babeczek za pomocą `resources.getQuantityString(R.plurals.cupcakes, numberOfCupcakes, numberOfCupcakes)`.

   Pełna `sendOrder()`metoda powinna wyglądać następująco:

```
fun sendOrder() {
   val numberOfCupcakes = sharedViewModel.quantity.value ?: 0
   val orderSummary = getString(
      R.string.order_details,
```

```
        resources.getQuantityString(R.plurals.cupcakes, numberOfCupcakes, numberOfCupcakes),
        sharedViewModel.flavor.value.toString(),
        sharedViewModel.date.value.toString(),
        sharedViewModel.price.value.toString()
    )

    val intent = Intent(Intent.ACTION_SEND)
        .setType("text/plain")
        .putExtra(Intent.EXTRA_SUBJECT, getString(R.string.new_cupcake_order))
        .putExtra(Intent.EXTRA_TEXT, orderSummary)

    if (activity?.packageManager?.resolveActivity(intent, 0) != null) {
        startActivity(intent)
    }
}
```

10. Uruchom i przetestuj swój kod. Sprawdź, czy podsumowanie zamówienia w treści wiadomości e-mail pokazuje 1 babeczkę w porównaniu z 6 babeczkami lub 12 babeczkami.

    Dzięki temu ukończyłeś wszystkie funkcje aplikacji Cupcake! Gratulacje!! To była z pewnością trudna aplikacja, a Ty poczyniłeś ogromne postępy na drodze do zostania programistą Androida! Udało Ci się z powodzeniem połączyć wszystkie pojęcia, których się do tej pory nauczyłeś, jednocześnie zdobywając nowe wskazówki dotyczące rozwiązywania problemów.

Ostatnie kroki

Teraz poświęć trochę czasu na uporządkowanie kodu, co jest dobrą praktyką kodowania, której nauczyłeś się z poprzednich ćwiczeń z programowania.

- Zoptymalizuj importy
- Sformatuj pliki
- Usuń nieużywany lub zakomentowany kod
- W razie potrzeby dodaj komentarze w kodzie

Aby Twoja aplikacja była bardziej dostępna, przetestuj ją z włączoną funkcją Talkback , aby zapewnić płynną obsługę. Komunikaty głosowe powinny pomóc w przekazaniu przeznaczenia każdego elementu na ekranie, tam gdzie jest to właściwe. Upewnij się również, że można przejść do wszystkich elementów aplikacji za pomocą gestów machnięcia.

Dokładnie sprawdź, czy wszystkie zaimplementowane przypadki użycia działają zgodnie z oczekiwaniami w końcowej aplikacji. Przykłady:

- Dane powinny być zachowane przy rotacji urządzenia (dzięki modelowi widoku).
- Jeśli dotkniesz przycisku W **górę** lub **Wstecz** , informacje o zamówieniu nadal powinny być wyświetlane poprawnie na `FlavorFragment`i `PickupFragment`.
- Wysłanie zamówienia do innej aplikacji powinno zawierać prawidłowe dane zamówienia.
- Anulowanie zamówienia powinno usunąć wszystkie informacje w zamówieniu.

Jeśli znajdziesz jakieś błędy, śmiało je napraw.

Dobra robota przy podwójnym sprawdzaniu swojej pracy!

6. Kod rozwiązania

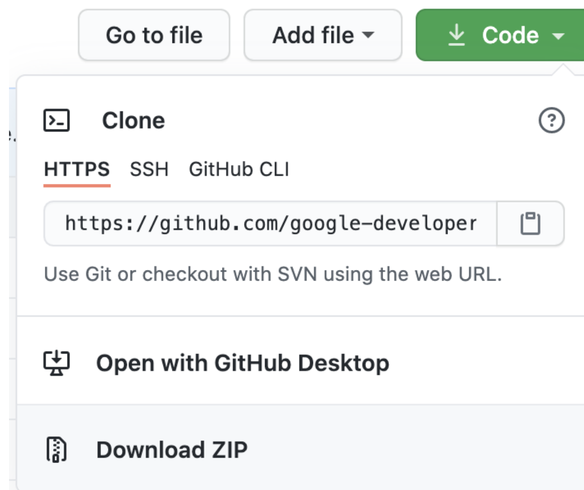
Kod rozwiązania dla tego ćwiczenia programowania znajduje się w projekcie pokazanym poniżej.

Adres URL kodu rozwiązania: https://github.com/google-developer-training/android-basics-kotlin-cupcake-app

Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

Pobierz kod

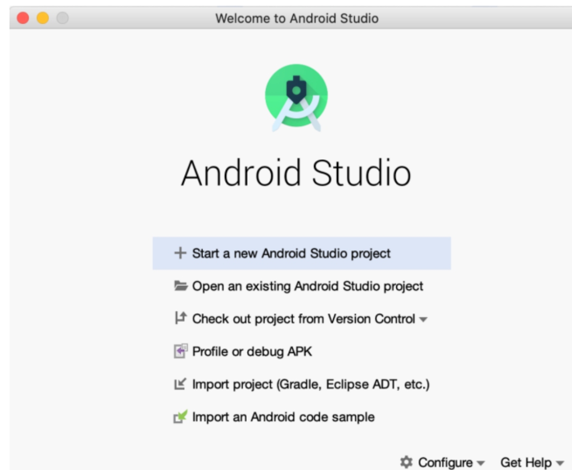
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



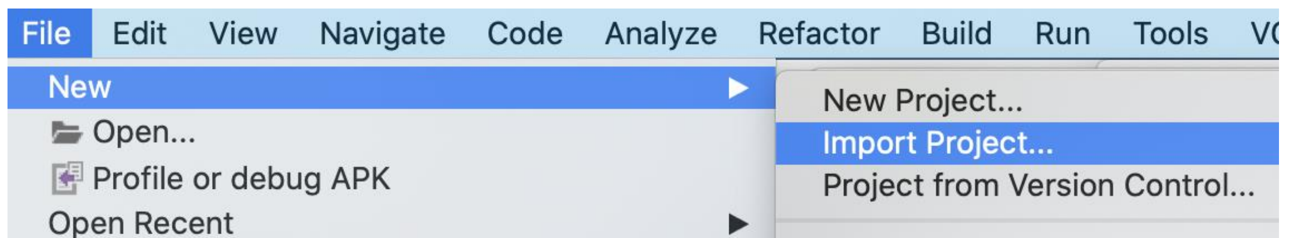
3. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
4. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
5. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

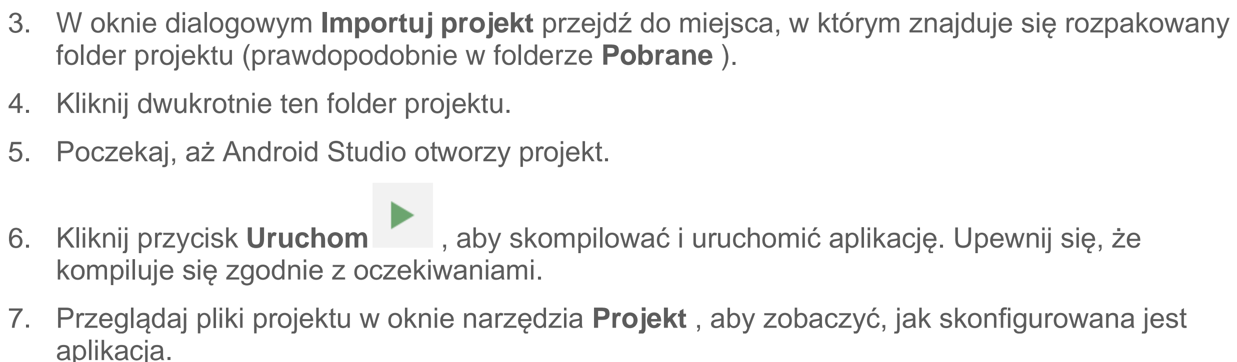
Otwórz projekt w Android Studio

1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz istniejący projekt Android Studio.**



Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Nowy** > **Importuj projekt** .





3. W oknie dialogowym **Importuj projekt** przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.
6. Kliknij przycisk **Uruchom** , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.
7. Przeglądaj pliki projektu w oknie narzędzia **Projekt** , aby zobaczyć, jak skonfigurowana jest aplikacja.

7. Podsumowanie

- Android przechowuje stos wszystkich odwiedzonych miejsc docelowych, a każdy nowy cel jest umieszczany w stosie.
- Naciskając przycisk W **górę** lub **Wstecz** , możesz usunąć miejsca docelowe z tylnego stosu.
- Korzystanie z komponentu Jetpack Navigation pomaga wypychać i usuwać miejsca docelowe fragmentów z tylnego stosu, dzięki czemu domyślne zachowanie przycisku **Wstecz** jest bezpłatne.
- Określ `app:popUpTo`atrybut w akcji na wykresie nawigacyjnym, aby usunąć miejsca docelowe z tylnego stosu do momentu określonego w wartości atrybutu.
- Określ `app:popUpToInclusive="true"`akcję, kiedy miejsce docelowe określone w `app:popUpTo`powinno również zostać usunięte z tylnego stosu.

- Możesz utworzyć niejawny zamiar udostępnienia treści w aplikacji poczty e-mail, używając Intent.ACTION_SENDi wypełniając takie dodatki, jak Intent.EXTRA_EMAIL, Intent.EXTRA_SUBJECTi Intent.EXTRA_TEXTżeby wymienić tylko kilka.
- Użyj pluralszasobu, jeśli chcesz używać różnych zasobów w postaci ciągów na podstawie ilości, na przykład w liczbie pojedynczej lub mnogiej.

8. Dowiedz się więcej

- Zasady nawigacji
- Zrozum zadania i tylny stos
- Nawigacja i tylny stos
- popUpTo i popUpToInclusive
- Zamiar e-maila
- Formatowanie ciągów
- Ilość ciągów (liczba mnoga)
- Operator Elvisa w Kotlinie

9. Ćwicz na własną rękę

Uwaga: Ćwiczenia są opcjonalne. Dają ci możliwość przećwiczenia tego, czego nauczyłeś się podczas tego ćwiczenia z programowania.

Rozszerz **aplikację Cupcake** o własne warianty procesu zamawiania babeczek. Przykłady:

- Zaoferuj specjalny smak, który wiąże się z pewnymi specjalnymi warunkami, takimi jak brak możliwości odbioru tego samego dnia.
- Zapytaj użytkownika o imię i nazwisko w zamówieniu babeczek.
- Pozwól użytkownikowi wybrać wiele smaków babeczek do zamówienia, jeśli ilość jest większa niż 1 babeczka.

Które obszary Twojej aplikacji musisz zaktualizować, aby uwzględnić tę nową funkcję?

Sprawdź swoją pracę:

Gotowa aplikacja powinna działać bez błędów.

10. Wyzwanie zadania

Uwaga: wszystkie wyzwania są opcjonalne i nie są wymagane do następnej aktywności. Używają i mogą kwestionować to, czego nauczyłeś się w tym i poprzednich powiązanych ćwiczeniach z programowania.

Wykorzystaj to, czego nauczyłeś się podczas tworzenia aplikacji **Cupcake** , aby zbudować aplikację do własnego przypadku użycia. Może to być aplikacja do zamawiania pizzy, kanapek lub czegokolwiek innego, o czym tylko pomyślisz! Zalecamy naszkicowanie różnych miejsc docelowych aplikacji przed rozpoczęciem jej wdrażania.

Aby czerpać inspirację z innych pomysłów projektowych, możesz również sprawdzić aplikację Shrine , która jest badaniem materiałów, które pokazuje, w jaki sposób możesz zastosować

motywy i komponenty materiałów dla własnej marki. Aplikacja Shrine jest znacznie bardziej złożona niż aplikacja **Cupcake** , którą zbudowałeś, więc zamiast dążyć do stworzenia bardzo wymagającej aplikacji z góry, zastanów się najpierw nad małymi funkcjami, z którymi możesz się uporać. Z biegiem czasu buduj swoją pewność siebie dzięki rosnącym i stopniowym wygranym.

Po zakończeniu tworzenia własnej aplikacji udostępnij to, co stworzyłeś w mediach społecznościowych. Użyj hashtagu #LearningKotlin, abyśmy mogli to zobaczyć!

Testuj modele widoków i dane na żywo

1. Zanim zaczniesz

W poprzednich ćwiczeniach z kodowania nauczyłeś się używać `ViewModel`s do obsługi logiki biznesowej, a także `LiveData`do reaktywnych interfejsów użytkownika. W tym laboratorium programowania nauczysz się pisać testy jednostkowe, aby sprawdzić, czy Twój `ViewModel`kod działa poprawnie.

Warunki wstępne

- Utworzyłeś katalogi testowe w Android Studio.
- Napisałeś testy jednostkowe i oprzyrządowanie w Android Studio.
- Do projektu Androida dodano zależności Gradle.

Czego się nauczysz

- Jak pisać testy jednostkowe dla `ViewModel`s i `LiveData`.

Czego potrzebujesz

- Komputer z zainstalowanym Android Studio.
- Kod rozwiązania dla aplikacji **Cupcake** .

Pobierz kod startowy do tego ćwiczenia z programowania

W tym ćwiczeniu z programowania dodasz testy oprzyrządowania do aplikacji **Cupcake** z poprzedniego kodu rozwiązania.

URL kodu startowego: https://github.com/google-developer-training/android-basics-kotlin-cupcake-app

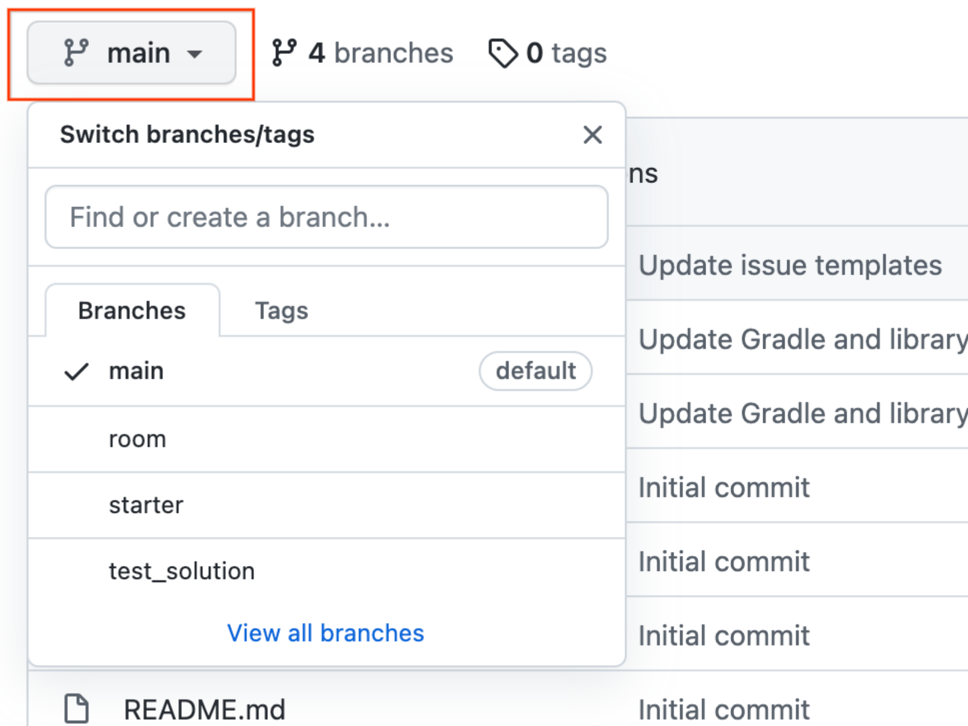
Nazwa modułu z kodem startowym:

`main`

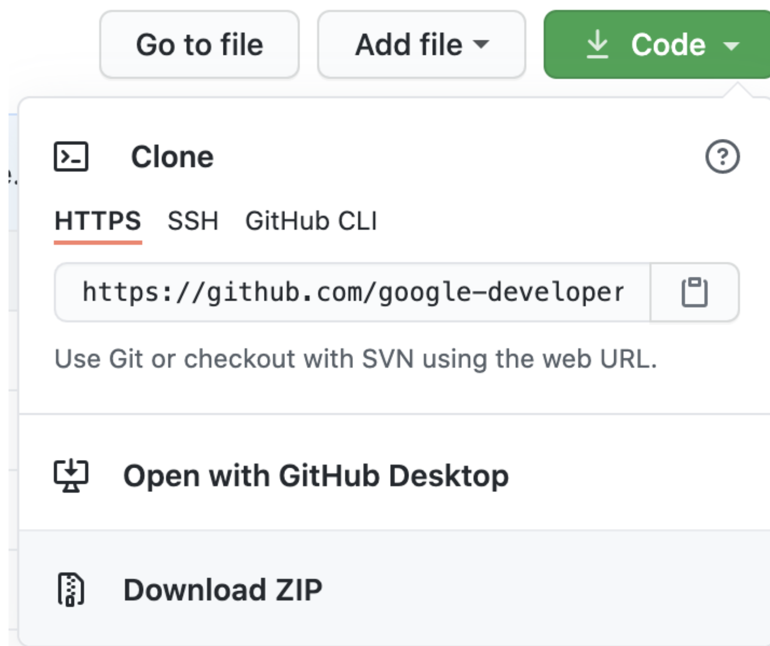
Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

Pobierz kod

1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Sprawdź i potwierdź, że nazwa oddziału jest zgodna z nazwą oddziału określoną w ćwiczeniach z programowania. Na przykład na poniższym zrzucie ekranu nazwa gałęzi to **main** .



3. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli wyskakujące okienko.

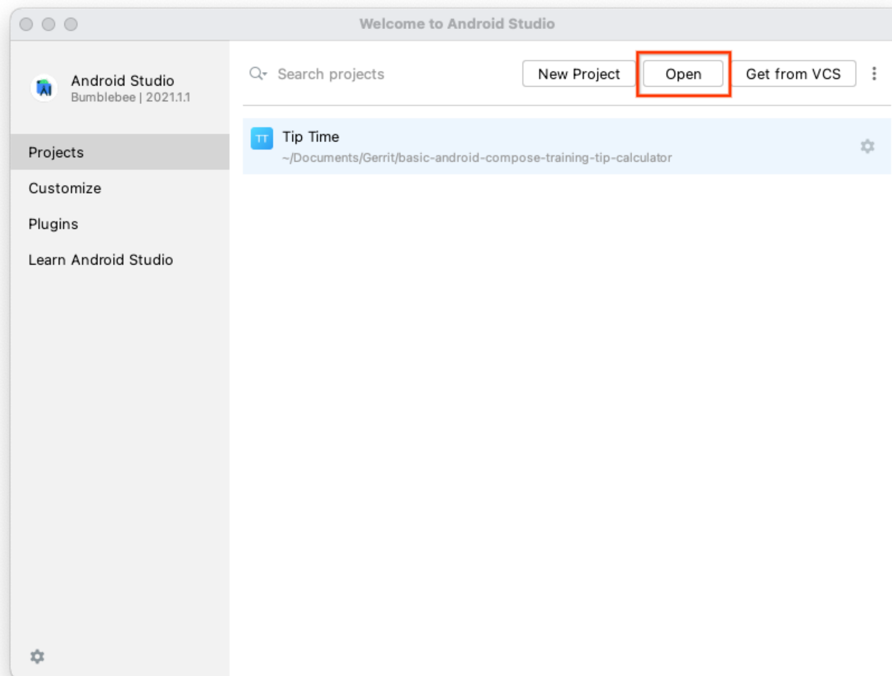


4. W wyskakującym okienku kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na komputerze. Poczekaj na zakończenie pobierania.
5. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
6. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

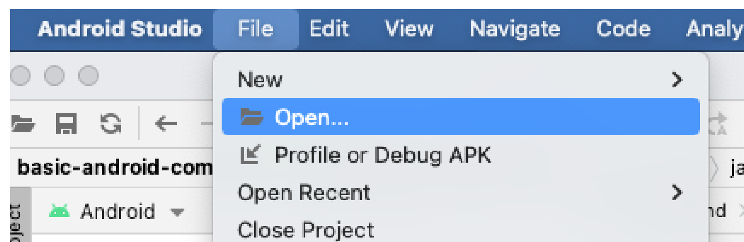
Otwórz projekt w Android Studio

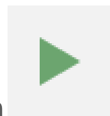
1. Uruchom Android Studio.

2. W oknie **Witamy w Android Studio** kliknij **Otwórz** .



Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Otwórz** .



3. W przeglądarce plików przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.
6. Kliknij przycisk **Uruchom**  , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.

2. Przegląd aplikacji startowej

Aplikacja **Cupcake** składa się z ekranu głównego, który pokazuje ekran zamówienia z trzema opcjami ilości babeczek. Kliknięcie opcji przeniesie Cię do ekranu, na którym wybierasz smak, a następnie przeniesie Cię do ekranu wyboru daty odbioru zamówienia. Następnie możesz wysłać zamówienie do innej aplikacji. Możesz anulować swoje zamówienie na dowolnym z tych etapów.

3. Utwórz katalogi testów jednostkowych

Utwórz katalog testów jednostkowych dla aplikacji Cupcake, tak jak robiłeś to w poprzednich ćwiczeniach z programowania.

4. Utwórz klasę testów jednostkowych

Utwórz nową klasę o nazwie **ViewModelTests.kt** .

5. Dodaj niezbędne zależności

Dodaj do swojego projektu następujące zależności:

```
testImplementation 'junit:junit:4.+'
testImplementation 'androidx.arch.core:core-testing:2.1.0'
```

Teraz zsynchronizuj swój projekt.

6. Napisz test ViewModel

Zacznijmy od prostego testu. Pierwszą rzeczą, którą robimy, gdy wchodzimy w interakcję z aplikacją na urządzeniu lub emulatorze, jest wybór ilości babeczek. Więc najpierw przetestujemy `setQuantity()`metodę w `OrderViewModel`, i sprawdzimy wartość `quantity` `LiveData`obiektu.

Zmienna `quantity`, którą będziemy testować, jest instancją `LiveData`. Testowanie `LiveData`obiektów wymaga dodatkowego kroku i właśnie w tym momencie wchodzi w grę dodana przez nas zależność. Używamy `LiveData`do aktualizacji naszego interfejsu użytkownika, gdy tylko zmieni się wartość. Nasz interfejs użytkownika działa na tym, co nazywamy „głównym wątkiem". Jeśli nie znasz się na wątkach i współbieżności, to w porządku, omówimy to szczegółowo w innych ćwiczeniach z kodowania. Na razie, w kontekście aplikacji na Androida, pomyśl o głównym wątku jako wątku interfejsu użytkownika. Kod, który pokazuje interfejs użytkownika, jest uruchamiany w tym wątku. O ile nie określono inaczej, test jednostkowy zakłada, że wszystko działa w wątku głównym. Jednak ponieważ `LiveData`obiekty nie mogą uzyskać dostępu do głównego wątku, musimy wyraźnie stwierdzić, że `LiveData`obiekty nie powinny wywoływać głównego wątku.

1. Aby określić, że `LiveData`obiekty nie powinny wywoływać głównego wątku, musimy podać konkretną regułę testową za każdym razem, gdy testujemy `LiveData`obiekt.

   ```
   @get:Rule
   var instantTaskExecutorRule = InstantTaskExecutorRule()
   ```

2. Teraz możemy stworzyć funkcję o nazwie `quantity_twelve_cupcakes()`. W metodzie utwórz wystąpienie`OrderViewModel.`

3. W tym teście będziesz sprawdzał, czy quantityobiekt w pliku OrderViewModeljest aktualizowany po setQuantitywywołaniu. Ale przed wywołaniem jakichkolwiek metod lub pracą z jakimikolwiek danymi w OrderViewModel, ważne jest, aby pamiętać, że podczas testowania wartości LiveDataobiektu, obiekty muszą być obserwowane w celu wyemitowania zmian. Prostym sposobem na zrobienie tego jest użycie observeForevermetody. Wywołaj observeForevermetodę na quantityobiekcie. Ta metoda wymaga wyrażenia lambda, ale można je pozostawić puste.
4. Następnie wywołaj setQuantity()metodę, przekazując 12ją jako parametr.

```
val viewModel = OrderViewModel()
viewModel.quantity.observeForever {}
viewModel.setQuantity(12)
```

5. Możemy śmiało wywnioskować, że wartość quantityobiektu to 12. Zauważ, że LiveDataobiekty nie są samą wartością. Wartości są zawarte we właściwości o nazwie value. Zrób następujące stwierdzenie:

```
assertEquals(12, viewModel.quantity.value)
```

Twój test powinien wyglądać tak:

```
@Test
fun quantity_twelve_cupcakes() {
  val viewModel = OrderViewModel()
  viewModel.quantity.observeForever {}
  viewModel.setQuantity(12)
  assertEquals(12, viewModel.quantity.value)
}
```

Uruchom swój test! Gratulacje, właśnie napisałeś swój pierwszy LiveDatatest jednostkowy, który jest kluczową umiejętnością we współczesnym rozwoju Androida. Ten test nie testuje zbyt wiele logiki biznesowej, więc napiszmy nieco bardziej skomplikowany test.

Jedną z głównych funkcji OrderViewModeljest obliczanie ceny naszego zamówienia. Dzieje się tak, gdy wybieramy ilość babeczek, a także kiedy wybieramy datę odbioru. Kalkulacja ceny odbywa się metodą prywatną, więc nasz test nie może bezpośrednio wywołać tej metody. Tylko inne metody w programie OrderViewModelmogą to wywołać. Te metody są publiczne, więc wywołamy je, aby uruchomić kalkulację ceny, abyśmy mogli sprawdzić, czy wartość ceny jest taka, jakiej oczekujemy.

Najlepsze praktyki

Cena jest aktualizowana po wybraniu ilości babeczek i po wybraniu daty. Chociaż oba te elementy powinny być przetestowane, generalnie preferowane jest testowanie tylko pod kątem pojedynczej funkcjonalności. Dlatego stworzymy oddzielne metody dla każdego testu: jedną funkcję do testowania ceny po aktualizacji ilości i oddzielną funkcję do testowania ceny po aktualizacji daty. Nigdy nie chcemy, aby wynik testu zakończył się niepowodzeniem, ponieważ nie powiódł się inny test.

1. Utwórz wywołaną metodę price_twelve_cupcakes()i opisz ją jako test.
2. W metodzie utwórz instancję OrderViewModeli wywołaj setQuantity()metodę, przekazując 12 jako parametr.

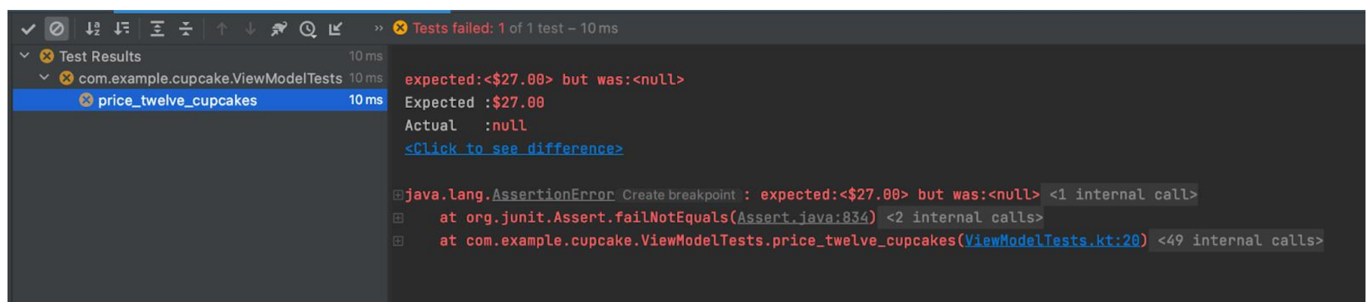
```
val viewModel = OrderViewModel()
viewModel.setQuantity(12)
```

3. Patrząc na PRICE_PER_CUPCAKEin OrderViewModel, widzimy, że babeczki kosztują 2,00 USD za sztukę. Widzimy również, że resetOrder()jest on wywoływany za każdym razem, gdy ViewModeljest inicjowany, a w tej metodzie domyślną datą jest dzisiejsza data i PRICE_FOR_SAME_DAY_PICKUPwynosi 3,00 USD. Zatem 12 * 2 + 3 = 27. Spodziewamy się, że wartość pricezmiennej po wybraniu 12 babeczek wyniesie 27,00 $. Załóżmy więc, że nasza wartość oczekiwana 27,00 USD jest równa wartości price LiveDataobiektu.

```
assertEquals("$27.00", viewModel.price.value)
```

Teraz uruchom test.

Powinno się nie udać!



Wynik testu mówi, że nasza rzeczywista wartość wynosiła null. Jest na to wytłumaczenie. Jeśli spojrzysz na pricezmienną w OrderViewModel, zobaczysz to:

```
val price: LiveData<String> = Transformations.map(_price) {
  // Format the price into the local currency and return this as LiveData<String>
  NumberFormat.getCurrencyInstance().format(it)
}
```

To jest przykład, dlaczego LiveDatawarto zwrócić uwagę na testowanie. Wartość pricejest ustawiana za pomocą Transformation. Zasadniczo ten kod pobiera wartość, którą przypisujemy, pricei przekształca ją w format walutowy, więc nie musimy tego robić ręcznie. Jednak ten kod ma inne konsekwencje. Podczas przekształcania LiveDataobiektu kod nie jest wywoływany, chyba że jest to absolutnie konieczne, co oszczędza zasoby na urządzeniu mobilnym. Kod zostanie wywołany tylko wtedy, gdy zaobserwujemy zmiany w obiekcie. Oczywiście odbywa się to w naszej aplikacji, ale musimy również zrobić to samo w przypadku testu.

4. W swojej metodzie testowej dodaj następujący wiersz przed ustawieniem ilości:

```
viewModel.price.observeForever { }
```

Twój test powinien wyglądać tak:

```
@Test
fun price_twelve_cupcakes() {
```

```
  val viewModel = OrderViewModel()
  viewModel.price.observeForever {}
  viewModel.setQuantity(12)
  assertEquals("$27.00", viewModel.price.value)
}
```

Teraz, jeśli uruchomisz test, powinien przejść.

7. Kod rozwiązania

URL kodu rozwiązania: https://github.com/google-developer-training/android-basics-kotlin-cupcake-app

Nazwa modułu z kodem rozwiązania:

test_solution

8. Gratulacje

W tym ćwiczeniu z programowania:

- Dowiedziałem się, jak skonfigurować `LiveData`test.
- Nauczył się `LiveData`sam się testować.
- Nauczyłem się, jak testować `LiveData`, że jest przekształcony.
- Nauczono się obserwować `LiveData`w teście jednostkowym.

Układy adaptacyjne

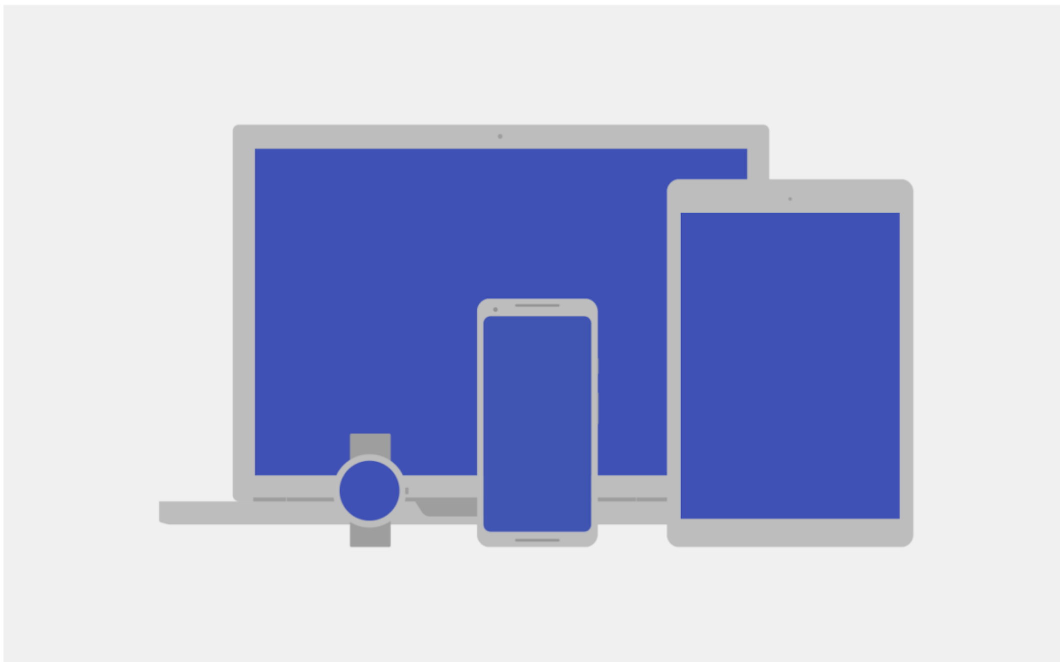
Dowiedz się, jak dostosować aplikacje do różnych rozmiarów ekranu.

Układy adaptacyjne

1. Zanim zaczniesz

Urządzenia z Androidem są dostępne w różnych kształtach, rozmiarach i formach. Powinieneś zaprojektować swoją aplikację tak, aby działała na różnych typach urządzeń, od urządzeń z małym ekranem po urządzenia z większym ekranem. Deweloperzy piszący aplikacje gotowe do produkcji mogą obsługiwać Android Wear , Android Auto i Android TV , ale te tematy są poza zakresem tego kursu. Gdy Twoja aplikacja obsługuje najszerszą gamę ekranów, możesz udostępnić ją jak największej liczbie użytkowników korzystających z różnych urządzeń.

Twoja aplikacja musi mieć elastyczny układ. Zamiast definiować układ za pomocą sztywnych wymiarów, które zakładają określone proporcje i rozmiar ekranu, układ powinien być w stanie zgrabnie dostosować się do różnych rozmiarów i orientacji ekranu. Ta sama zasada obowiązuje, gdy Twoja aplikacja działa na składanym urządzeniu, na którym rozmiar ekranu i proporcje mogą się zmieniać podczas działania aplikacji. Pod koniec tego ćwiczenia z kodowania nauczysz się krótkiego wprowadzenia do urządzeń składanych.



Warunki wstępne

- Jak pobrać kod do Android Studio i uruchomić go.
- Zaznajomiony z komponentami architektury Androida `ViewModel`i `LiveData`.
- Podstawowa znajomość komponentów nawigacyjnych.

Czego się nauczysz

- Jak dodać `SlidingPaneLayout`do swojej aplikacji.

Co zbudujesz

- Zaktualizuj aplikację Sports, aby dostosować ją do dużych ekranów.

Co będziesz potrzebował

- Komputer z zainstalowanym Android Studio.
- Kod startowy do aplikacji **Sports** .

Pobierz kod startowy do tego ćwiczenia z programowania

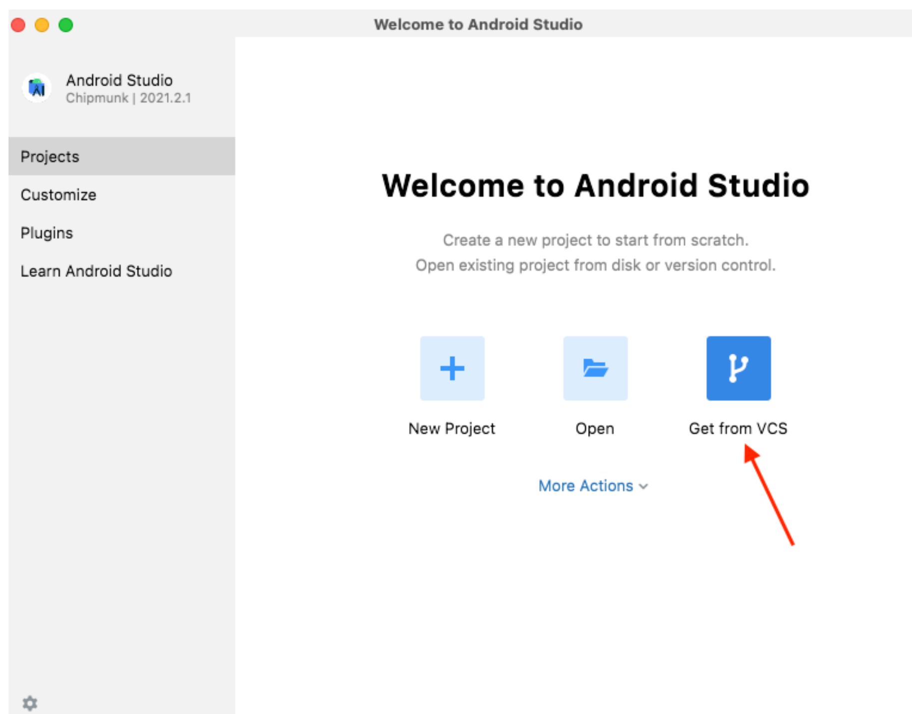
To ćwiczenie z programowania zapewnia kod startowy, który można rozszerzyć o funkcje nauczane w tym ćwiczeniu z programowania. Kod startowy może zawierać kod, który jest Ci znany z poprzednich ćwiczeń z programowania, a także kod, który jest Ci nieznany, o którym dowiesz się w późniejszych ćwiczeniach z programowania.

URL kodu startowego: https://github.com/google-developer-training/basic-android-kotlin-training-sports/tree/starter

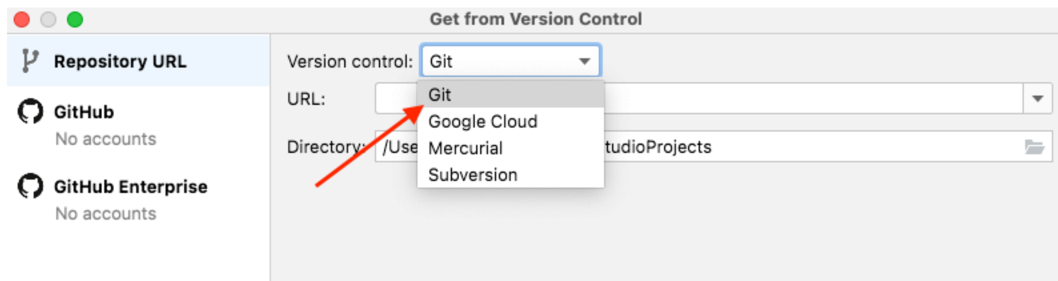
Nazwa oddziału z kodem startowym:`starter`

Aby pobrać kod tego ćwiczenia z programowania z GitHub i otworzyć go w Android Studio, wykonaj następujące czynności.

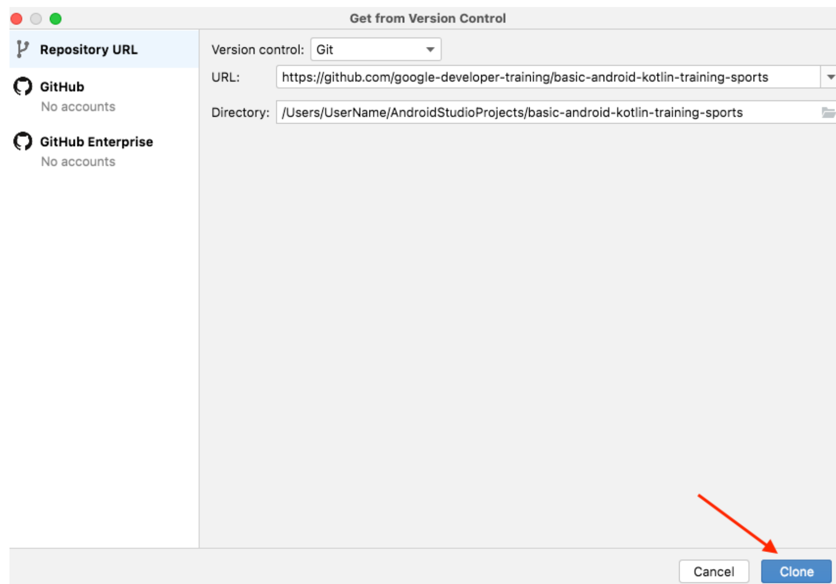
1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Pobierz z VCS** .



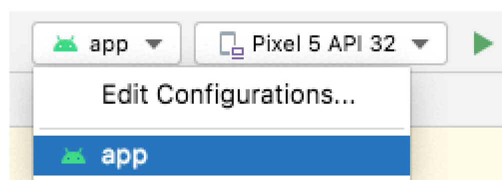
3. W oknie dialogowym **Pobierz z kontroli wersji** upewnij się, że **Git** jest wybrany do **kontroli wersji** .



4. Wklej podany adres URL kodu w polu **adresu URL** .
5. Opcjonalnie zmień **katalog** na inny niż sugerowany domyślny.



6. Kliknij **Klonuj** . Android Studio rozpocznie pobieranie Twojego kodu.
7. Poczekaj, aż otworzy się Android Studio.
8. Wybierz odpowiedni moduł dla swojego startera, aplikacji lub kodu rozwiązania.



9. Kliknij przycisk **Uruchom** , aby skompilować i uruchomić swój kod.

2. Obejrzyj wideo z kodem (opcjonalnie)

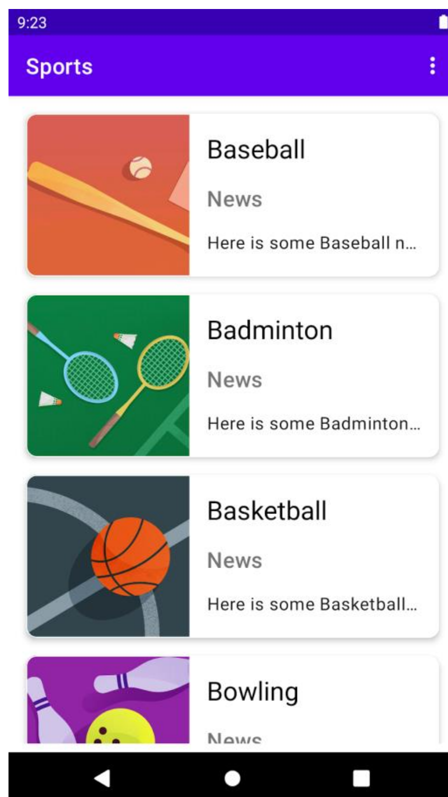
Jeśli chcesz zobaczyć, jak jeden z instruktorów kursu wykonuje ćwiczenia z programowania, obejrzyj poniższy film.

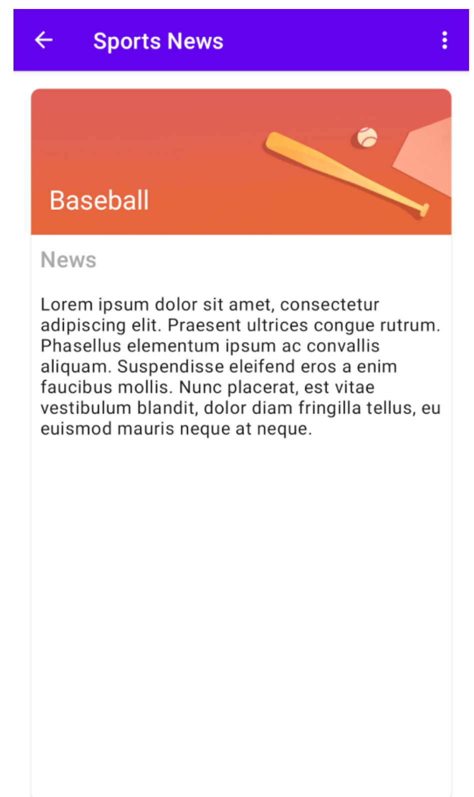
Zaleca się rozwinięcie wideo do pełnego ekranu (z tą ikoną w prawym dolnym rogu wideo), aby lepiej widzieć Android Studio i kod.

Ten krok jest opcjonalny. Możesz także pominąć wideo i od razu rozpocząć instrukcje ćwiczeń z programowania.

3. Przegląd aplikacji startowej

Aplikacja sportowa składa się z dwóch ekranów. Pierwszy ekran wyświetla listę sportów. Użytkownik może wybrać konkretną pozycję sportową i wyświetlany jest drugi ekran. Drugi ekran to ekran szczegółów, na którym wyświetlane są wybrane wiadomości sportowe. Ekran szczegółów wyświetla tekst zastępczy, aby uprościć implementację.



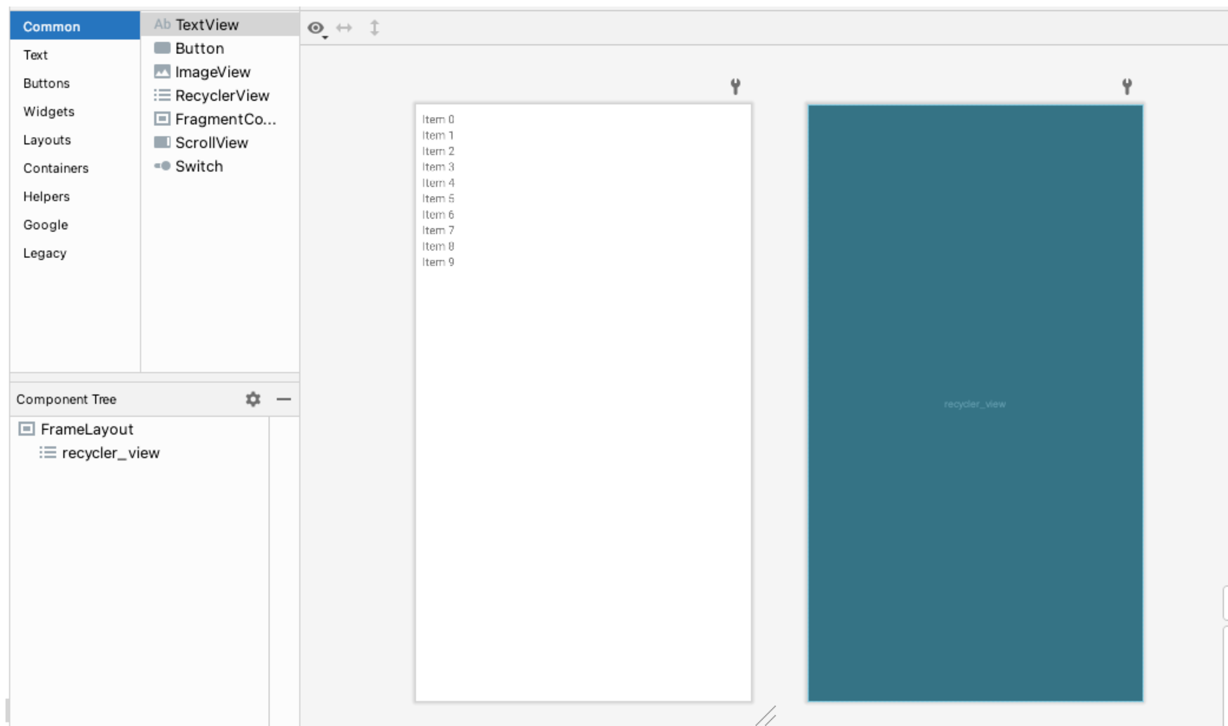


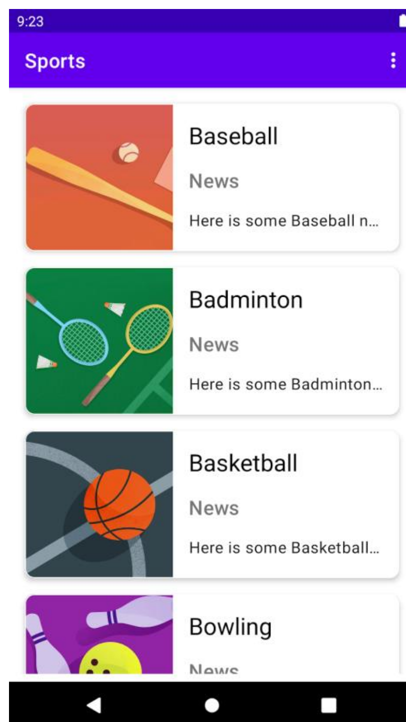
Przejdź przez kod startowy

Pobrany kod startowy zawiera ekran listy i układy ekranów szczegółów wstępnie zaprojektowane dla Ciebie. W tej ścieżce skupisz się tylko na dostosowaniu swojej aplikacji do dużych ekranów. Użyjesz `SlidingPaneLayout`, aby skorzystać z dużego ekranu. Oto krótki opis niektórych plików na początek.

fragment_sports_list.xml

- Otwórz res/layout/fragment_sports_list.xmlw widoku **Projekt** .
- Zawiera układ pierwszego ekranu w Twojej aplikacji, czyli listy sportowej.
- Ten układ składa się z Recyclerview, który pokazuje listę wiadomości sportowych.





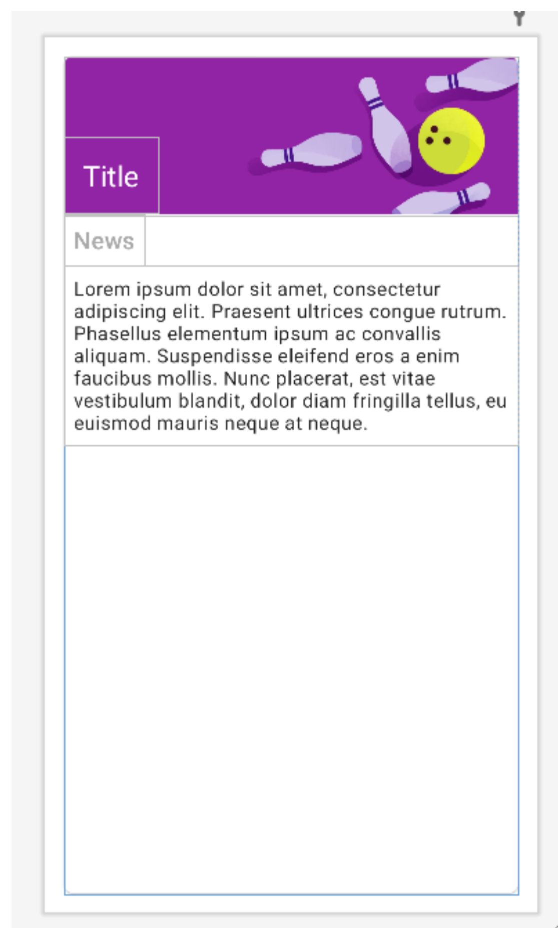
sports_list_item.xml

- Otwórz res/layout/sports_list_item.xmlw widoku **Projekt** .
- Zawiera układ każdego elementu w widoku Recyclerview.

- Ten układ składa się z miniatury obrazu sportowego, tytułu Wiadomości i tekstu zastępczego dla krótkich wiadomości sportowych.

fragment_sports_news.xml

- Otwórz res/layout/fragment_sports_news.xml w widoku **Projekt** .
- Zawiera układ drugiego ekranu w Twojej aplikacji. Ten ekran jest wyświetlany, gdy użytkownik wybierze sport z widoku Recyclerview.
- Ten układ składa się ze sportowego baneru graficznego i tekstu zastępczego dla wiadomości sportowych.



main_activity.xml i content_main.xml

Te dwa definiują główny układ działań za pomocą jednego fragmentu.

nawigacja/nav_graph.xml

Wykres nawigacji zawiera dwa miejsca docelowe, jeden z listami sportowymi, a drugi z wiadomościami sportowymi.

folder res/values

Znasz pliki zasobów w tym folderze.

- colors.xmlzawiera kolory motywu używane w aplikacji.
- strings.xmlzawiera wszystkie ciągi znaków, których potrzebuje Twoja aplikacja.
- themes.xmlzawierają dostosowanie interfejsu użytkownika wykonane dla Twojej aplikacji.

MainActivity.kt

Zawiera domyślny kod wygenerowany przez szablon, aby ustawić widok treści działania jako main_activity.xml. Metoda onSupportNavigateUp()jest zastępowana, aby obsługiwać domyślną nawigację w górę z paska aplikacji.

model/Sport.kt

Jest to klasa danych, która przechowuje dane, które mają być wyświetlane w każdym wierszu listy sportowej Recyclerview.

data/SportsData.kt

Ten plik zawiera funkcję o nazwie getSportsData(), która zwraca ArrayListwstępnie wypełnione zakodowane dane sportowe.

Ostrzeżenie : nie jest zalecaną praktyką zakodowanie danych w kodzie na stałe. Aby zachować prostotę i skupić się na układach adaptacyjnych, w tej aplikacji ciągi są zakodowane na stałe.

SportsViewModel.kt

To jest wspólne ViewModeldla aplikacji. ViewModelWspólny jest pierwszy SportsListFragmentekran z listą sportów i NewsDetailsFragmentdrugi ekran ze szczegółowymi wiadomościami sportowymi.

- Właściwość _currentSportjest typu MutableLiveData,, który przechowuje aktualny sport wybrany przez użytkownika. Właściwość currentSportjest właściwością zapasową dla _currentSporti jest udostępniana jako publiczna wersja tylko do odczytu dla innych klas.
- Nieruchomość _sportsDatazawiera listę danych sportowych. Podobnie jak w przypadku poprzedniej właściwości, sportsDatajest publiczną wersją tylko do odczytu tej właściwości.
- Blok inicjatora init{}inicjuje _currentSporti _sportsData. Jest _sportsDatainicjowany z całą listą sportów z data/SportsData.kt. Jest _currentSportinicjowany z pierwszą pozycją na liście.
- Funkcja updateCurrentSport()przyjmuje Sportsinstancję i aktualizuje _currentSportprzekazaną wartość.

SportsAdapter.kt

To jest adapter do RecyclerView. W konstruktorze przekazywany jest detektor kliknięć. Większość kodu w tym pliku to standardowy kod, który znasz z poprzednich ćwiczeń z programowania.

SportsListFragment.kt

To pierwszy fragment ekranu, na którym wyświetlana jest lista sportów.

- onCreateView()Funkcja rozszerza kod fragment_sports_listXML układu przy użyciu obiektu powiązania.
- onViewCreated()funkcja konfiguruje RecyclerViewadapter. Aktualizuje wybrany przez użytkownika sport jako aktualny sport w udostępnionym ViewModel. SportsViewModelPrzechodzi do ekranu szczegółów z wiadomościami sportowymi i przesyła listę sportów do adaptera w celu wyświetlenia za pomocą submitList(List).

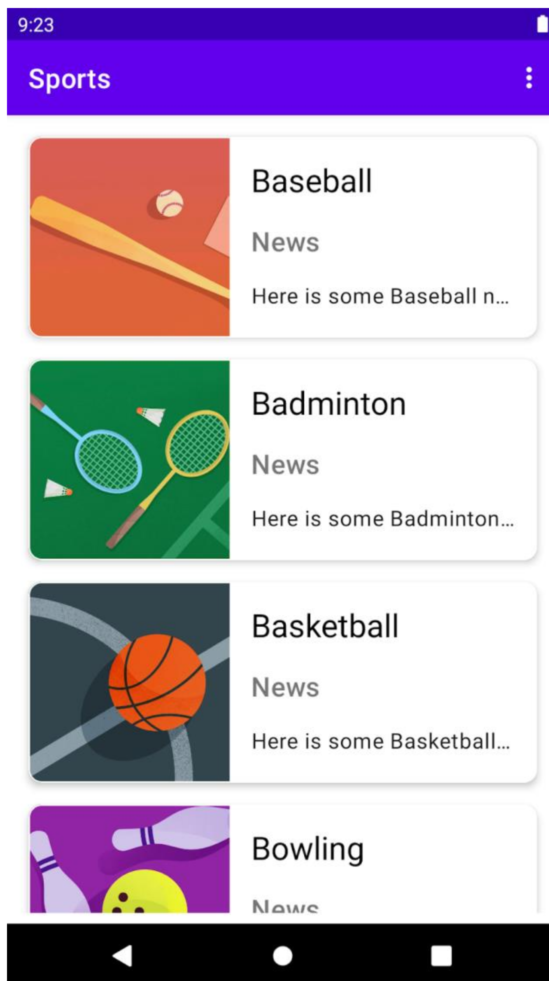
AktualnościSzczegółyFragment.kt

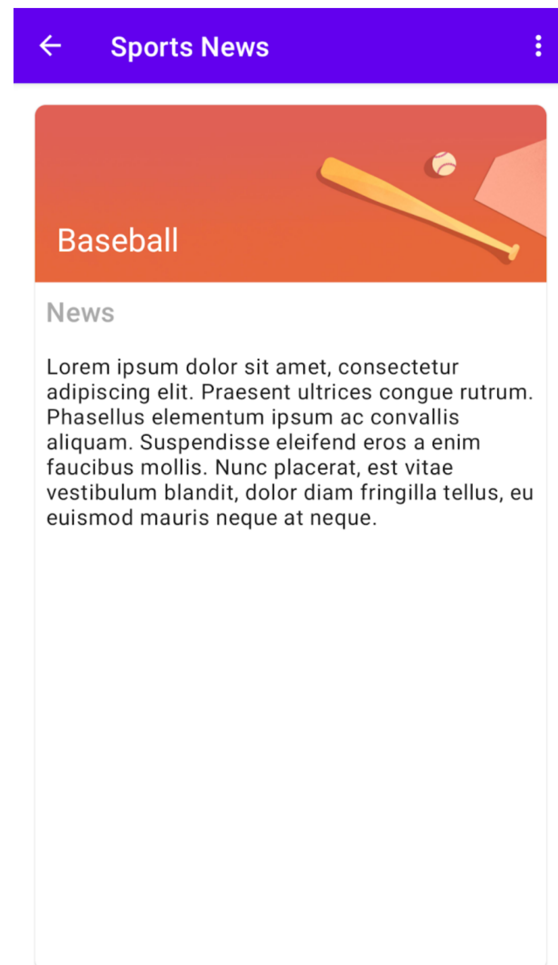
To jest drugi ekran w Twojej aplikacji, na którym wyświetlany jest tekst zastępczy dla wiadomości sportowych.

- onCreateView()Funkcja rozszerza kod fragment_sports_newsXML układu przy użyciu obiektu powiązania.
- onViewCreated()funkcja dołącza obserwatora do właściwości SportsViewModel's, currentSport aby automatycznie aktualizować interfejs użytkownika po zmianie danych. Wewnątrz obserwatora aktualizowany jest tytuł sportowy, wizerunek i wiadomości.

Zbuduj i uruchom aplikację

1. Zbuduj i uruchom aplikację na emulatorze lub urządzeniu. Wybierz dowolny element z listy sportowej, a aplikacja powinna przejść do drugiego ekranu z tekstem zastępczym dla wiadomości.





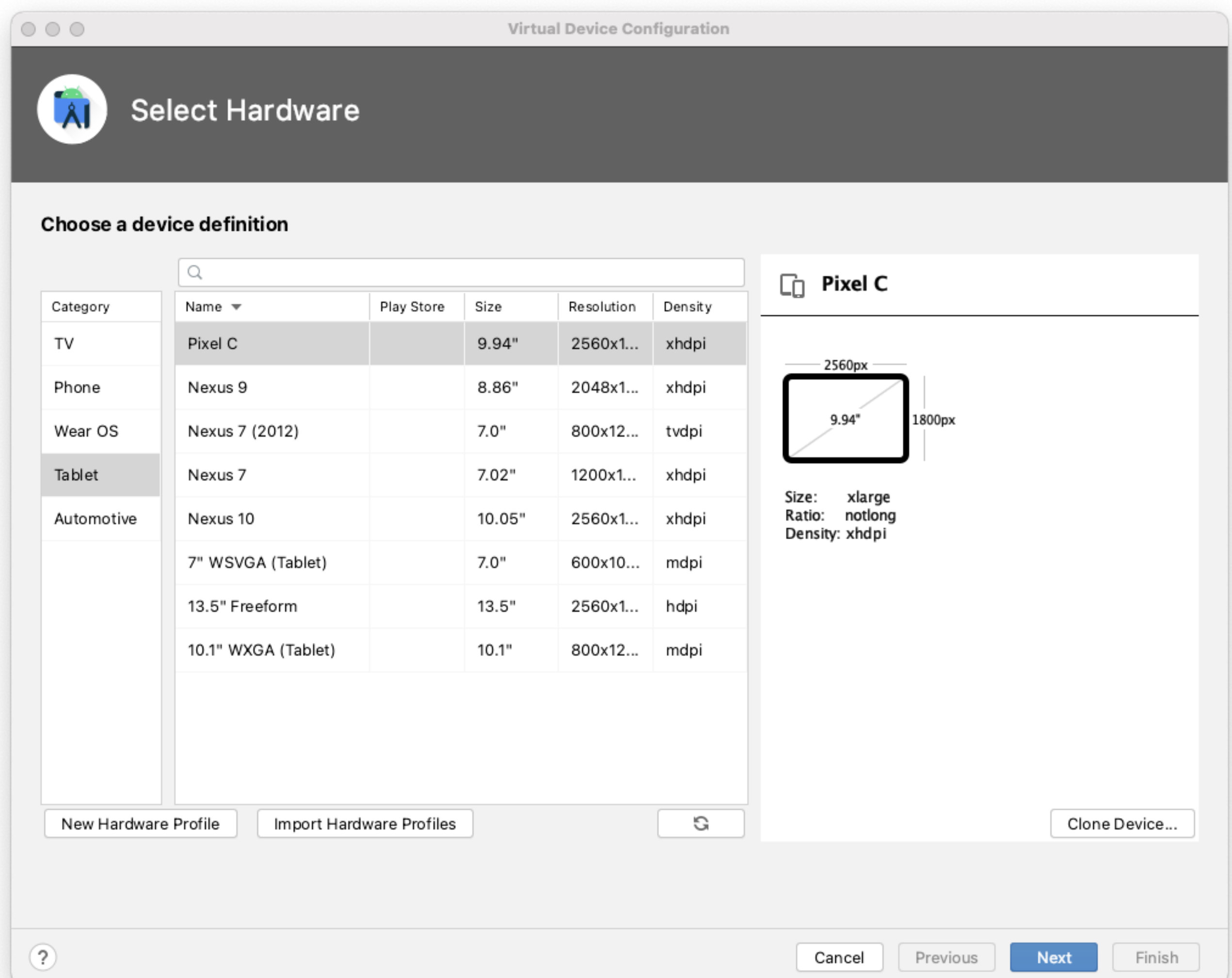
4. Wzór listy-szczegółów

Obecna aplikacja startowa nie wykorzystuje w pełni możliwości ekranu na większych urządzeniach, takich jak tablety. Aby rozwiązać ten problem, wyświetlisz interfejs użytkownika aplikacji przy użyciu wzorca List-Detail, którego nauczysz się w tym laboratorium.

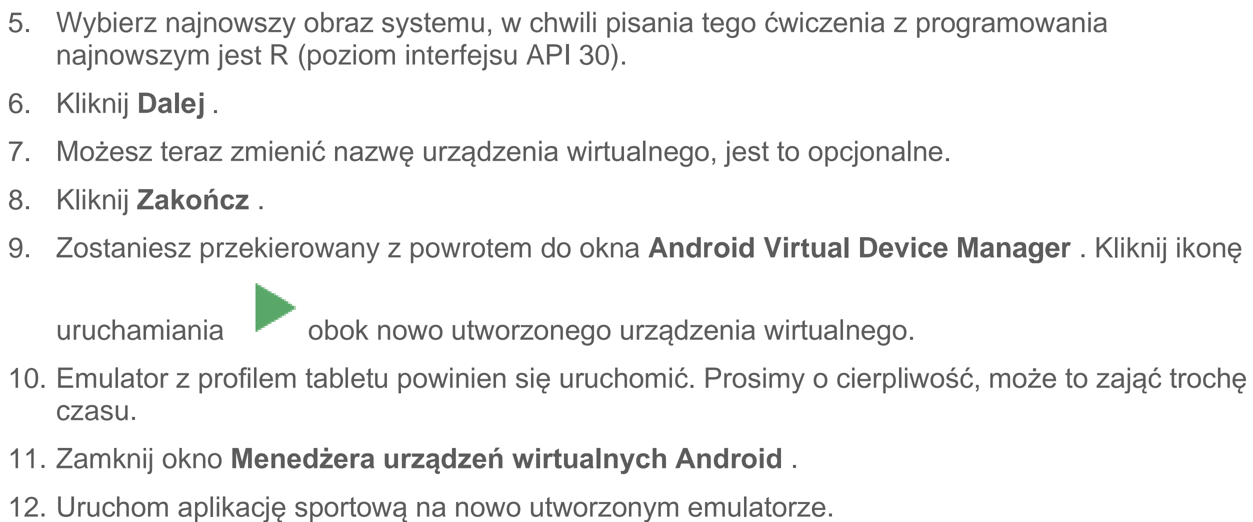
Uruchom aplikację na tablecie

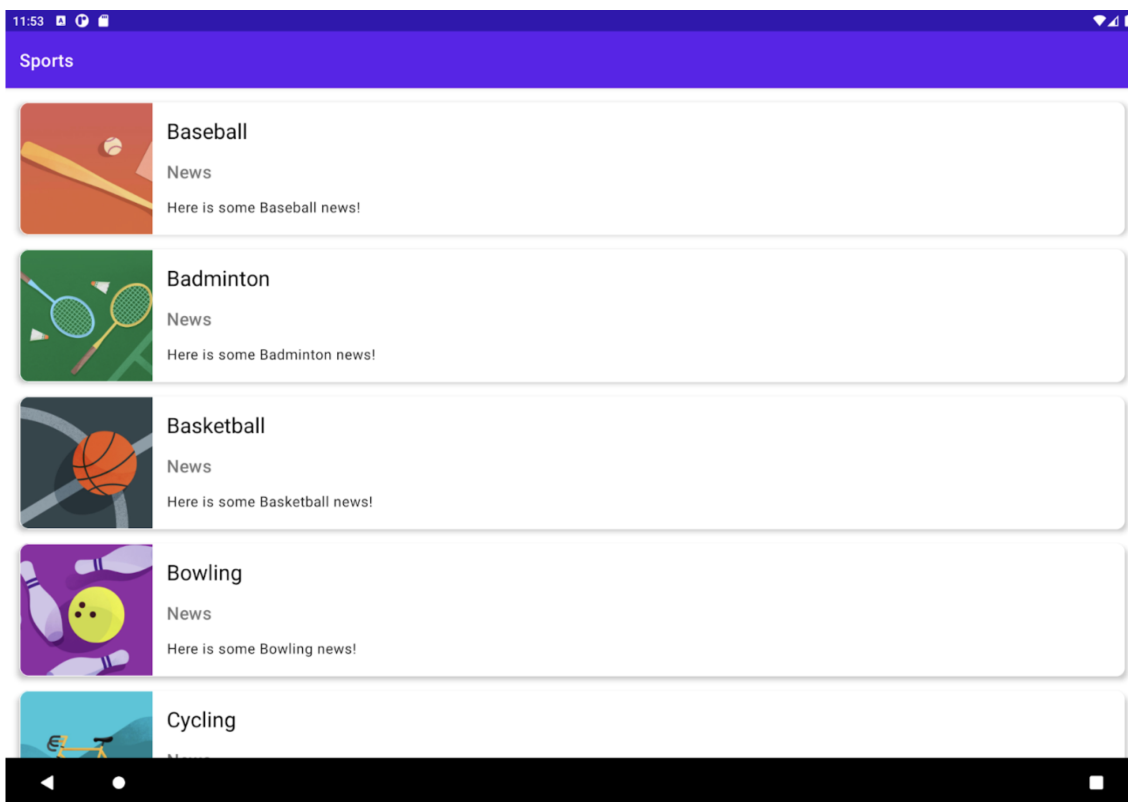
W tym zadaniu stworzysz emulator z profilem tabletu. Po utworzeniu emulatora uruchomisz kod startowy aplikacji sportowej i będziesz obserwować interfejs użytkownika.

1. W Android Studio przejdź do **Narzędzia** > **Menedżer AVD** .
2. Zostanie wyświetlone okno **Menedżera urządzeń wirtualnych Android .** Kliknij **+ Utwórz nowe urządzenie wirtualne...** wyświetlane na dole.
3. Wyświetlone zostanie okno **Konfiguracja urządzenia wirtualnego .** Tutaj skonfigurujesz sprzęt i system operacyjny emulatora. Kliknij **Tablet** w lewym okienku. Wybierz **Pixel C** lub inny podobny profil sprzętowy w środkowym okienku.



4. Kliknij **Dalej** .

5. Wybierz najnowszy obraz systemu, w chwili pisania tego ćwiczenia z programowania najnowszym jest R (poziom interfejsu API 30).
6. Kliknij **Dalej** .
7. Możesz teraz zmienić nazwę urządzenia wirtualnego, jest to opcjonalne.
8. Kliknij **Zakończ** .
9. Zostaniesz przekierowany z powrotem do okna **Android Virtual Device Manager** . Kliknij ikonę uruchamiania  obok nowo utworzonego urządzenia wirtualnego.
10. Emulator z profilem tabletu powinien się uruchomić. Prosimy o cierpliwość, może to zająć trochę czasu.
11. Zamknij okno **Menedżera urządzeń wirtualnych Android** .
12. Uruchom aplikację sportową na nowo utworzonym emulatorze.

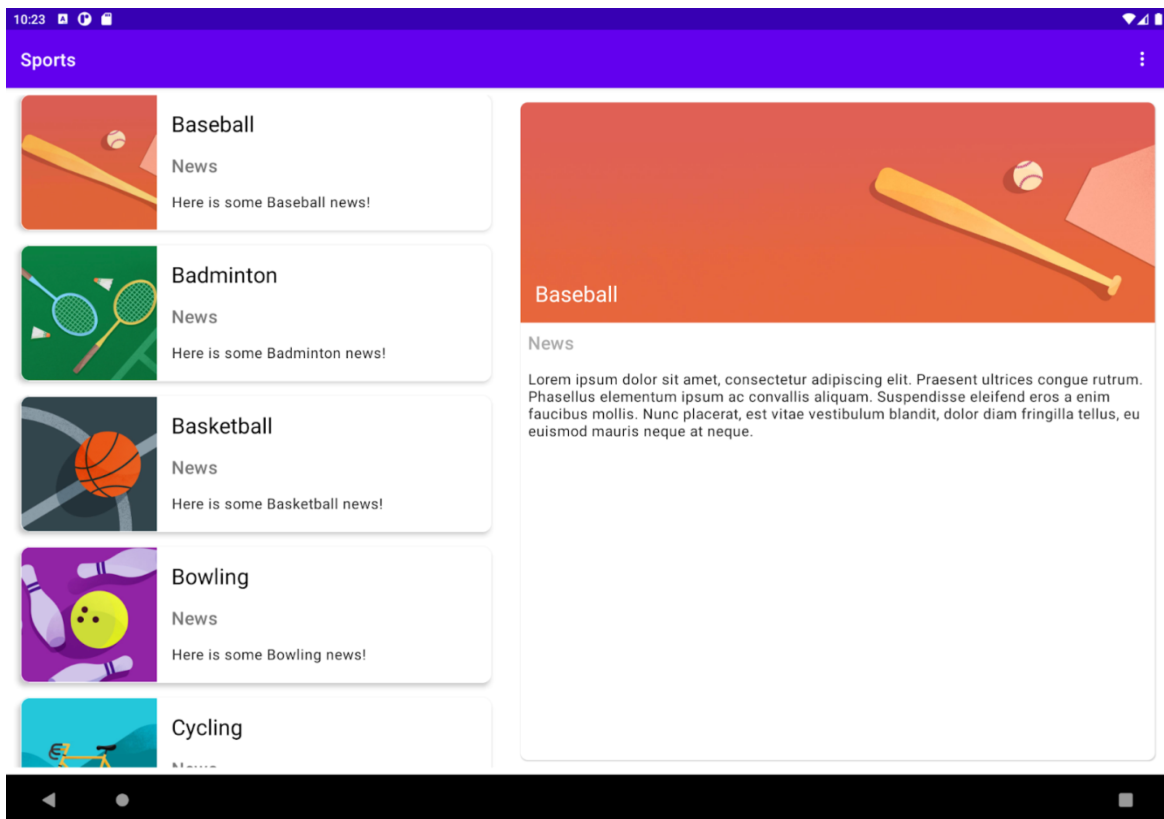


Uwaga na dużych urządzeniach aplikacja nie wykorzystuje całego ekranu. Szczegółowa lista jest bardziej efektywna na dużym ekranie niż lista. Wzorzec element-szczegół, zwany również wzorcem główny-szczegółowy, przedstawia listę elementów po jednej stronie układu, a po dotknięciu elementu obok niego są wyświetlane szczegóły. Zazwyczaj widoki te są wyświetlane tylko na dużych ekranach, takich jak tablety, ponieważ mają więcej miejsca na wyświetlenie większej ilości treści.

Poniższe obrazy są przykładem wzorca listy szczegółów:

Powyższe wzorce szczegółów listy wyświetlają listę elementów po lewej stronie i szczegóły wybranej pozycji po prawej stronie.

W ten sam sposób, jeśli użyjesz powyższego wzoru w swojej aplikacji sportowej, fragment wiadomości będzie Twoim ekranem szczegółów.



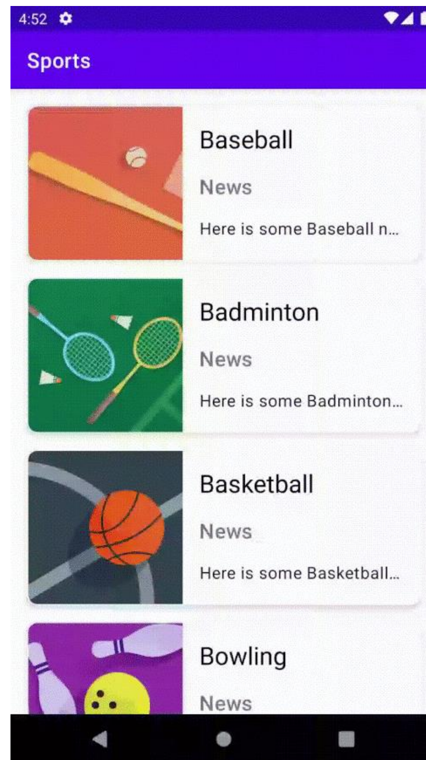
W tym ćwiczeniu z programowania dowiesz się, jak zaimplementować interfejs użytkownika list-detail za pomocą `SlidingPaneLayout`.

5. Wzór układu przesuwnego panelu

Interfejs użytkownika z listą może wymagać innego zachowania w zależności od rozmiaru ekranu. Na dużych wyświetlaczach jest wystarczająco dużo miejsca, aby lista i panele szczegółów znajdowały się obok siebie. Kliknięcie elementu listy wyświetla jego szczegóły w

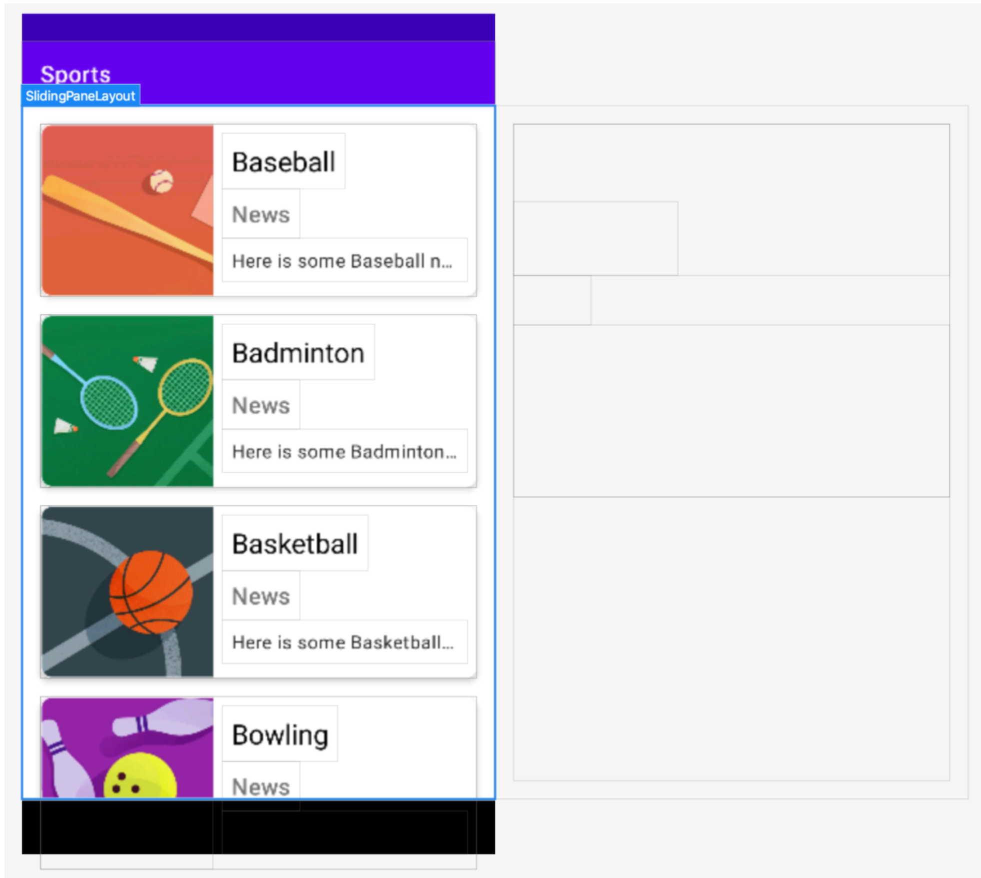
okienku szczegółów. Jednak na małych ekranach wydają się one zatłoczone. Zamiast wyświetlać oba panele jednocześnie, lepiej wyświetlać je pojedynczo. Początkowo panel listy wypełnia ekran. Dotknięcie elementu zastępuje panel listy panelem szczegółów tego elementu, który również wypełnia ekran.

Dowiesz się, jak używać SlidingPaneLayout do zarządzania logiką wyboru odpowiedniego środowiska użytkownika na podstawie bieżącego rozmiaru ekranu.



Zwróć uwagę, jak okienko szczegółów przesuwa się po okienku listy na mniejszych ekranach.

Poniżej znajdują się obrazy, które ilustrują, jak SlidingPaneLayoutwygląda na mniejszym ekranie. Obserwuj, jak okienko szczegółów nakłada się na okienko listy po wybraniu elementu z listy. Więc obie szyby są zawsze obecne!


Sports
SlidingPaneLayout
Baseball
News
Here is some Baseball n...
Badminton
News
Here is some Badminton...
Basketball
News
Here is some Basketball...
Bowling
News

Sports
SlidingPaneLayout
Basketball
News
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent ultrices congue rutrum. Phasellus elementum ipsum ac convallis aliquam. Suspendisse eleifend eros a enim faucibus mollis. Nunc placerat, est vitae vestibulum blandit, dolor diam fringilla tellus, eu euismod mauris neque at neque.

Dlatego SlidingPaneLayoutpodpory wyświetlają dwa panele obok siebie na większych urządzeniach, a jednocześnie automatycznie dostosowują się do wyświetlania tylko jednego panelu naraz na mniejszych urządzeniach, takich jak telefony.

6. Dodaj zależności biblioteczne

1. Otwórz build.gradle (Module: Sports.app).
2. W dependenciessekcji uwzględnij następującą zależność do użycia SlidingPaneLayoutw swojej aplikacji.

```
dependencies {
...
   implementation "androidx.slidingpanelayout:slidingpanelayout:1.2.0-beta01"
}
```

Uwaga : pobierz najnowsze numery wersji ze strony z wydaniami systemu AndroidX .

7. Skonfiguruj fragment listy sportowej xml

W tym zadaniu przekonwertujesz układ główny fragment_sports_listna SlidingPaneLayout. Jak już wiesz, SlidingPaneLayoutzapewnia poziomy układ z dwoma panelami do użytku na najwyższym poziomie interfejsu użytkownika. Ten układ używa pierwszego okienka jako listy zawartości lub przeglądarki podporządkowanej głównemu widokowi szczegółów do wyświetlania zawartości w drugim okienku.

W aplikacji Sports pierwszy panel będzie RecyclerViewwyświetlał listę sportów, a drugi panel wyświetla wiadomości sportowe.

Dodaj układ panelu przesuwnego

1. Otwórz fragment_sports_list.xml. Zauważ, że układ główny to FrameLayout.
2. Zmień FrameLayoutnaandroidx.slidingpanelayout.widget.SlidingPaneLayout.

```
<androidx.slidingpanelayout.widget.SlidingPaneLayout
  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"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".SportsListFragment">

  <androidx.recyclerview.widget.RecyclerView...>
</androidx.slidingpanelayout.widget.SlidingPaneLayout>
```

3. Dodaj android:idatrybut do SlidingPaneLayouti nadaj mu wartość @+id/sliding_pane_layout.

```
<androidx.slidingpanelayout.widget.SlidingPaneLayout
   ...
   android:id="@+id/sliding_pane_layout"
   ...>
```

Dodaj drugi panel do SlidingPaneLayout

W tym zadaniu dodasz drugie dziecko do `SlidingPaneLayout`. Będzie to wyświetlane jako prawy panel treści.

1. W `fragment_sports_list.xml`, poniżej `RecyclerView`, dodaj drugie dziecko, `androidx.fragment.app.FragmentContainerView`.
2. Dodaj wymagane atrybuty `layout_height`i `layout_width`do `FragmentContainerView`. Daj im wartość `match_parent`. Pamiętaj, że zaktualizujesz te wartości później.

```
<androidx.fragment.app.FragmentContainerView
   android:layout_height="match_parent"
   android:layout_width="match_parent"/>
```

3. Dodaj `android:id`atrybut do `FragmentContainerView`i nadaj mu wartość `@+id/detail_container`.

```
android:id="@+id/detail_container"
```

4. Dodaj `NewsDetailsFragment`do `FragmentContainerView`za pomocą `android:name`atrybutu.

```
android:name="com.example.android.sports.NewsDetailsFragment"
```

Zaktualizuj atrybut szerokość_układu

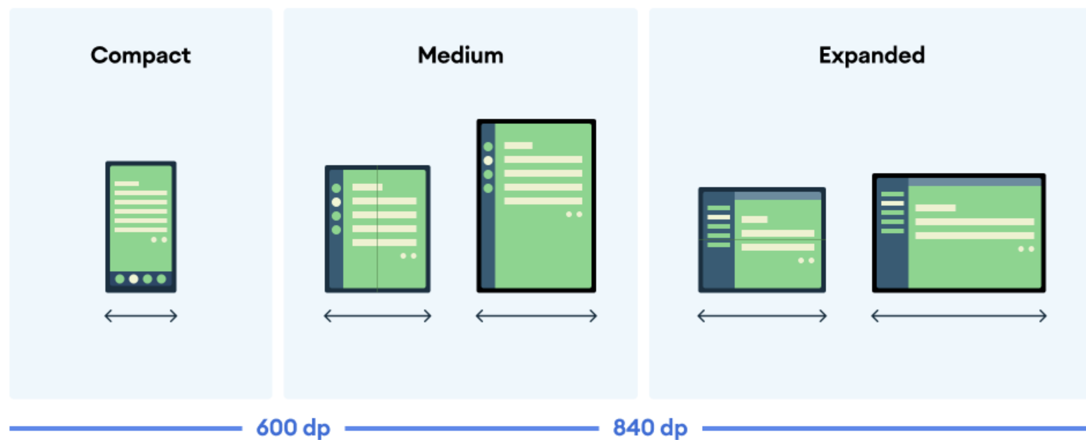
Używa `SlidingPaneLayout`szerokości dwóch paneli do określenia, czy wyświetlać panele obok siebie. Na przykład, jeśli mierzy się, że okienko listy ma minimalny rozmiar, `300dp`a okienko szczegółów wymaga `400dp`, to `SlidingPaneLayout`automatycznie wyświetla dwa okienka obok siebie, o ile ma co najmniej `700dp`dostępną szerokość.

Widoki podrzędne nakładają się na siebie, jeśli ich łączna szerokość przekracza dostępną szerokość w `SlidingPaneLayout`. W takim przypadku widoki podrzędne rozszerzają się, aby wypełnić dostępną szerokość w `SlidingPaneLayout`.

Aby określić szerokość widoków podrzędnych, powinieneś mieć kilka podstawowych informacji o szerokości ekranu urządzenia. W poniższej tabeli przedstawiono listę opiniowanych punktów przerwania, które należy zaprojektować, opracować i przetestować pod kątem układów aplikacji o zmiennym rozmiarze. Zostały one wybrane specjalnie, aby zrównoważyć prostotę układu z elastycznością optymalizacji aplikacji pod kątem wyjątkowych przypadków.

Szerokość	**Punkt przerwania**	**Reprezentacja urządzenia**
Kompaktowa szerokość	<600dp	99,96% telefonów w orientacji pionowej

Średnia szerokość	600dp+	93,73% tabletów w pionieDuże, rozłożone wyświetlacze wewnętrzne w pionie
Rozszerzona szerokość	840dp+	97,22% tabletów w orientacji poziomejDuże, rozłożone wyświetlacze wewnętrz orientacji poziomej

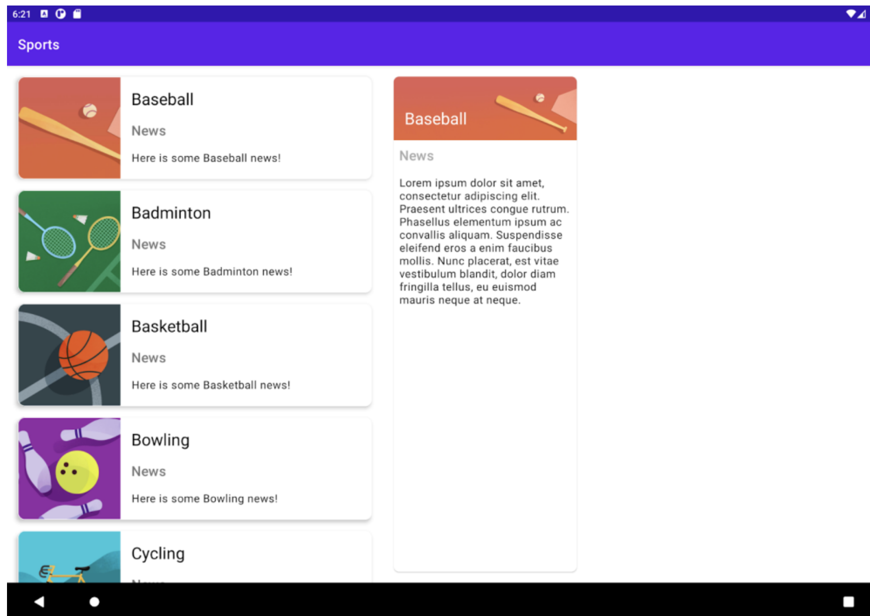


W aplikacji **Sports** chcesz wyświetlić pojedynczy panel, listę sportów na telefonach, czyli dla urządzeń o szerokości mniejszej niż 600dp. Aby wyświetlić oba okienka na tabletach, łączna szerokość powinna być większa niż 840dp. Możesz użyć szerokości 550dpdla pierwszego dziecka, widoku recyklera, a 300dpdla drugiego dziecka, FragmentContainerView.

1. W fragment_sports_list.xmlprogramie zmień szerokość układu RecyclerViewdo 550dpi FragmentContainerViewdo 300dp.

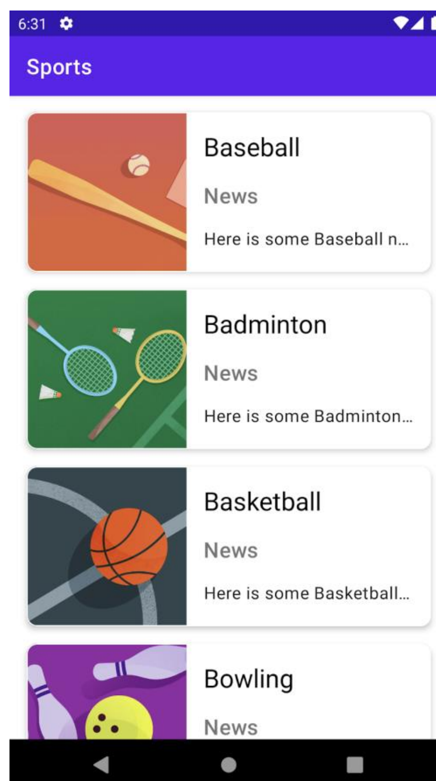
```
<androidx.recyclerview.widget.RecyclerView
  ...
  android:layout_width="550dp"
  .../>

<androidx.fragment.app.FragmentContainerView
  ...
  android:layout_width="300dp"
  .../>
```

2. Uruchom aplikację na emulatorze z profilem tabletu i emulatorem z profilem telefonu.



Zwróć uwagę, że na tablecie wyświetlane są dwa okienka. Szerokość drugiego okienka na tablecie zostanie ustalona na późniejszym etapie.

3. Uruchom aplikację na emulatorze z profilem telefonu.



Dodaj wagę układu

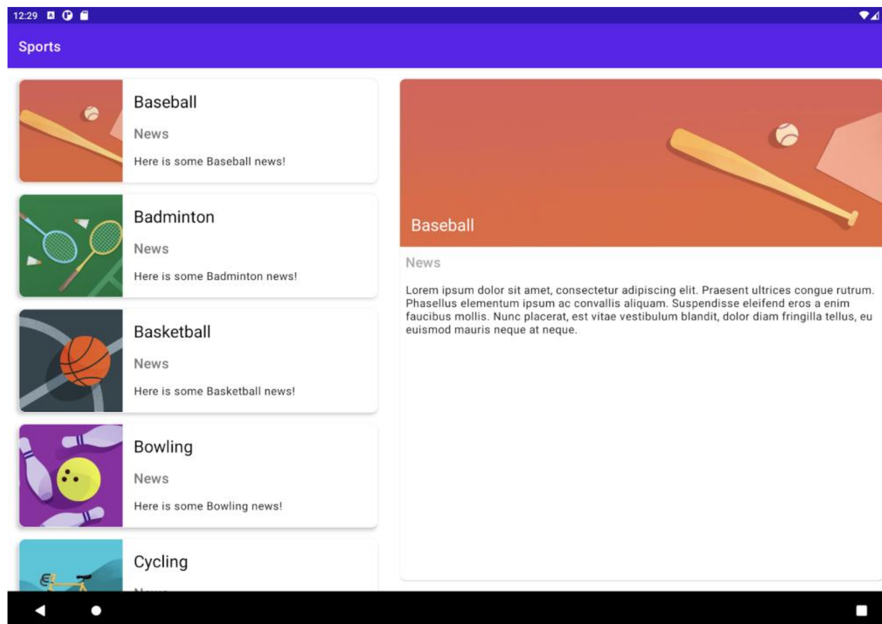
W tym zadaniu naprawisz interfejs użytkownika na tablecie i sprawisz, że drugie okienko zajmie całą pozostałą przestrzeń.

Podpory SlidingPaneLayoutdefiniujące sposób podziału pozostałej przestrzeni po pomiarze za pomocą parametru układu layout_weightw widokach podrzędnych, jeśli widoki się nie nakładają. Ten parametr dotyczy tylko szerokości.

1. W fragment_sports_list.xmlprogramie dodaj layout_weightdo FragmentContainerViewi nadaj mu wartość 1. Teraz drugi panel rozszerza się, aby wypełnić miejsce pozostałe po zmierzeniu panelu listy.

```
android:layout_weight="1"
```

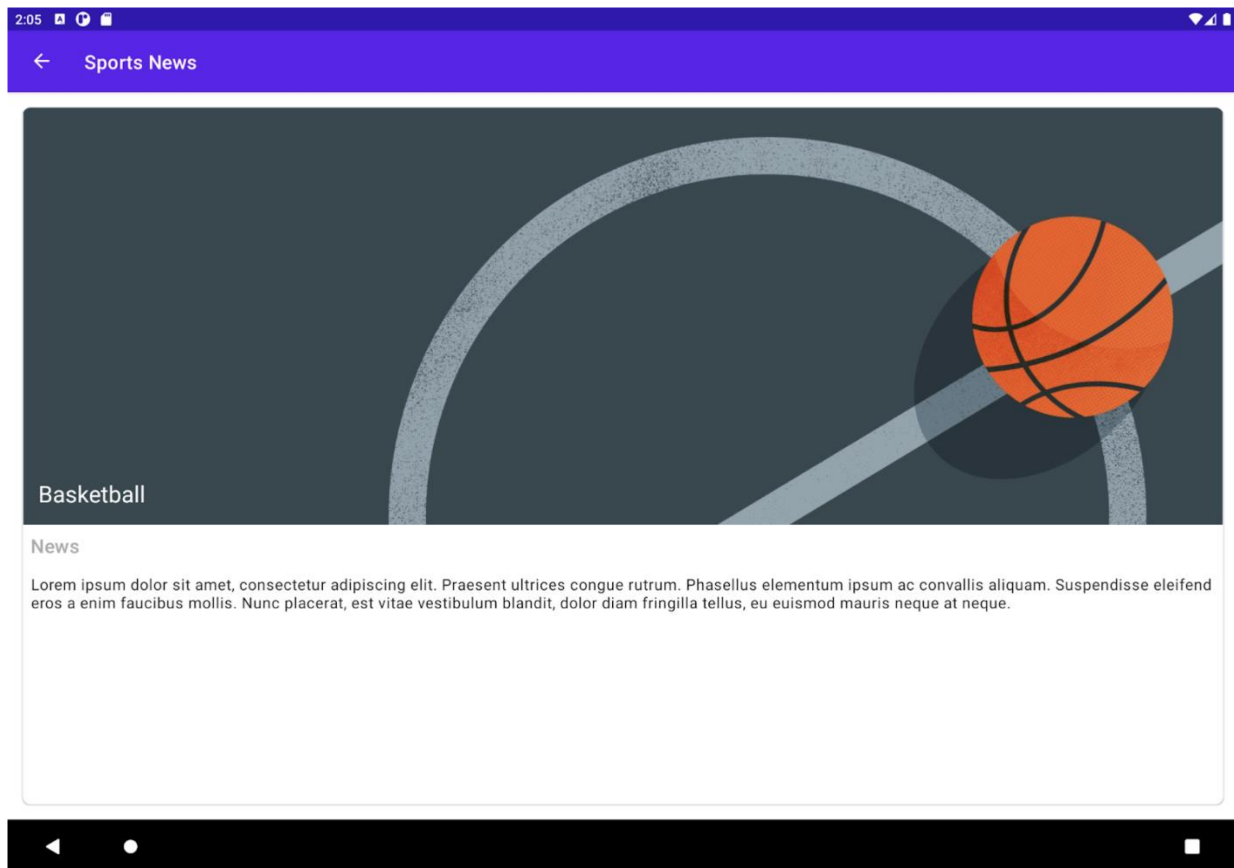
2. Uruchom aplikację.



Gratulacje! Pomyślnie dodałeś SlidingPaneLayout. Jeszcze nie skończyłeś. Musisz zaimplementować nawigację wsteczną i zaktualizować drugi panel, gdy element zostanie wybrany z listy. Zaimplementujesz je w późniejszym zadaniu.

8. Zamień okienko szczegółów

Uruchom aplikację na emulatorze z profilem tabletu. Wybierz pozycję z listy sportów. Zauważ, że aplikacja przechodzi do okienka szczegółów.



W tym zadaniu naprawisz ten problem. Obecnie zawartość podwójnego okienka jest aktualizowana o wybrany sport, a następnie aplikacja przechodzi do `NewsDetailsFragment`.

1. W `SportsListFragment`pliku, w funkcji `onViewCreated()`, zlokalizuj następujące wiersze, które prowadzą do ekranu szczegółów.

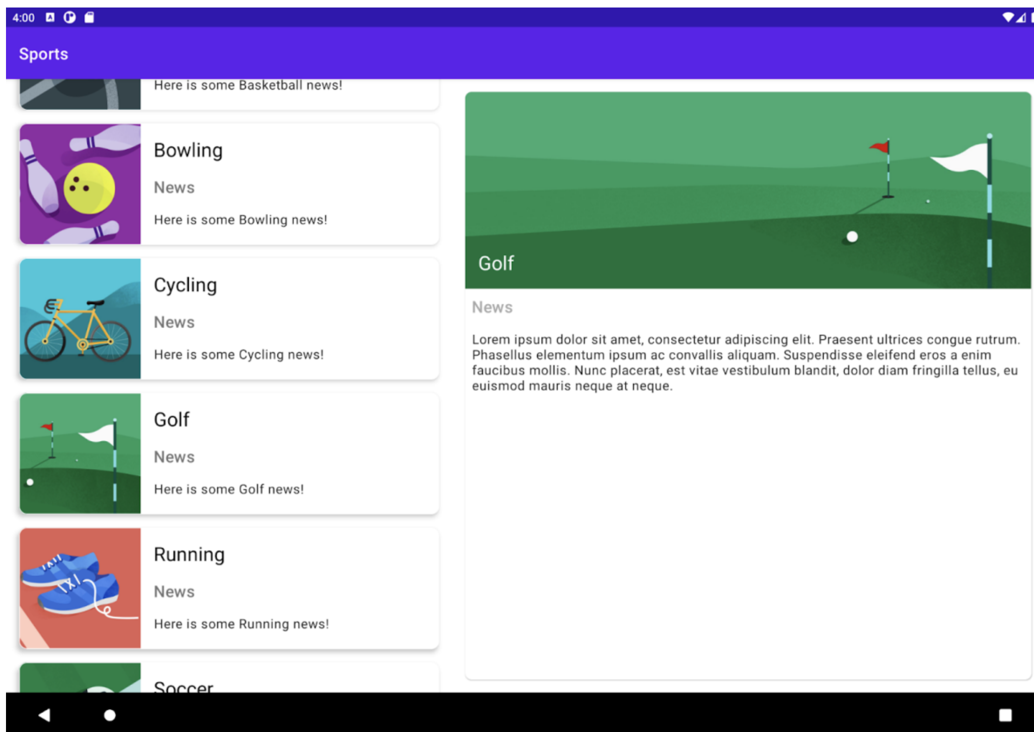
```
// Navigate to the details screen
val action = SportsListFragmentDirections.actionSportsListFragmentToNewsFragment()
this.findNavController().navigate(action)
```

2. Zastąp powyższe wiersze następującym kodem:

```
binding.slidingPaneLayout.openPane()
```

   Wywołaj `openPane()`, `SlidingPaneLayout` aby zamienić drugie okienko na pierwsze. Nie będzie to miało żadnego widocznego efektu, jeśli oba panele będą widoczne, np. na tablecie.

3. Uruchom aplikację na emulatorze tabletu i telefonu. Zwróć uwagę, że zawartość podwójnego okienka jest prawidłowo aktualizowana.



W następnym zadaniu następną funkcją, którą dodasz do aplikacji, jest niestandardowa nawigacja wsteczna.

9. Dodaj niestandardową nawigację wsteczną

W przypadku mniejszych urządzeń, w których panele listy i szczegółów nakładają się na siebie, należy upewnić się, że systemowy przycisk Wstecz przenosi użytkownika z powrotem z panelu szczegółów z powrotem do panelu listy. Możesz to zrobić, zapewniając niestandardową nawigację wsteczną i łącząc się `OnBackPressedCallback` z bieżącym stanem `SlidingPaneLayout`.

Powrót nawigacja

Nawigacja wsteczna to sposób, w jaki użytkownicy poruszają się wstecz po historii odwiedzonych wcześniej ekranów. Wszystkie urządzenia z systemem Android mają przycisk Wstecz do tego typu nawigacji. W zależności od urządzenia z systemem Android użytkownika ten przycisk może być przyciskiem fizycznym lub przyciskiem oprogramowania.

Niestandardowa nawigacja wsteczna

Android utrzymuje *tylny stos* miejsc docelowych, gdy użytkownik nawiguje po aplikacji. Zwykle pozwala to systemowi Android na prawidłowe nawigowanie do poprzednich miejsc docelowych po naciśnięciu przycisku Wstecz. Istnieje jednak kilka przypadków, w których aplikacja może wymagać zaimplementowania własnego zachowania Wstecz, aby zapewnić jak najlepsze wrażenia użytkownika.

Na przykład podczas korzystania z WebView, takiego jak przeglądarka Chrome, możesz chcieć zastąpić domyślne zachowanie przycisku Wstecz, aby umożliwić użytkownikowi nawigację wstecz po historii przeglądania sieci zamiast poprzednich ekranów w Twojej aplikacji.

Podobnie musisz zapewnić niestandardową nawigację wsteczną do SlidingPaneLayouti nawigować po aplikacji z okienka szczegółów z powrotem do okienka listy.

Zaimplementuj niestandardową nawigację wsteczną

Aby wdrożyć niestandardową nawigację wsteczną w aplikacji Sports, musisz:

- Zdefiniuj niestandardowe wywołanie zwrotne do obsługi naciśnięcia klawisza Wstecz, które zastępuje OnBackPressedCallback.
- Zarejestruj się i dodaj instancję wywołania zwrotnego.

Najpierw zdefiniuj niestandardowe wywołanie zwrotne.

1. W SportsListFragmentpliku dodaj nową klasę poniżej SportsListFragmentdefinicji klasy. Nazwij to SportsListOnBackPressedCallback.
2. Przekaż privateinstancję SlidingPaneLayoutjako parametr konstruktora.

```
class SportsListOnBackPressedCallback(
  private val slidingPaneLayout: SlidingPaneLayout
)
```

3. Rozszerz klasę z OnBackPressedCallback. Klasa OnBackPressedCallbackobsługuje onBackPressedwywołania zwrotne. Wkrótce naprawisz błąd parametru konstruktora.

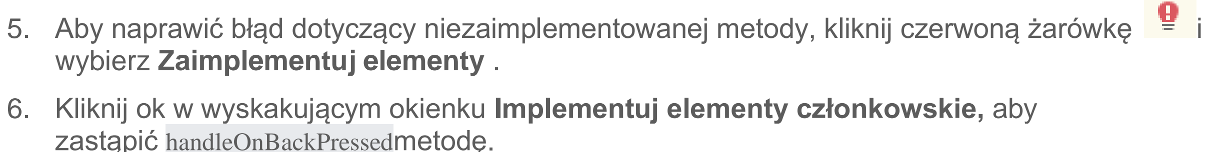
```
class SportsListOnBackPressedCallback(
  private val slidingPaneLayout: SlidingPaneLayout
): OnBackPressedCallback()
```

Konstruktor for OnBackPressedCallbackprzyjmuje wartość logiczną dla początkowego stanu włączenia. Tylko wtedy, gdy wywołanie zwrotne jest włączone (tj. isEnabled()zwraca true), dyspozytor wywoła wywołanie zwrotne, handleOnBackPressed()aby obsłużyć zdarzenie przycisku Wstecz.

4. Przekaż ** *jako parametr konstruktora do . Wartość logiczna* slidingPaneLayout.*isSlideable&& slidingPaneLayout.isOpenOnBackPressedCallbackisSlideable*będzie prawdziwe tylko wtedy, gdy drugie okienko jest przesuwane, co byłoby na mniejszym ekranie i wyświetlany jest pojedynczy okienko. Wartość *isOpen*będzie truejeśli drugi panel - panel zawartości jest całkowicie otwarty.

```
class SportsListOnBackPressedCallback(
  private val slidingPaneLayout: SlidingPaneLayout
): OnBackPressedCallback(slidingPaneLayout.isSlideable && slidingPaneLayout.isOpen)
```

Ten kod zapewni, że wywołanie zwrotne będzie włączone tylko na urządzeniach z mniejszym ekranem i gdy okienko zawartości jest otwarte.



5. Aby naprawić błąd dotyczący niezaimplementowanej metody, kliknij czerwoną żarówkę i wybierz **Zaimplementuj elementy** .
6. Kliknij ok w wyskakującym okienku **Implementuj elementy członkowskie,** aby zastąpić handleOnBackPressedmetodę.

Twoja klasa powinna wyglądać tak:

```
class SportsListOnBackPressedCallback(
  private val slidingPaneLayout: SlidingPaneLayout
): OnBackPressedCallback(slidingPaneLayout.isSlideable && slidingPaneLayout.isOpen) {
  /**
   * Callback for handling the [OnBackPressedDispatcher.onBackPressed] event.
   */
  override fun handleOnBackPressed() {
     TODO("Not yet implemented")
  }
}
```

7. Wewnątrz handleOnBackPressed()funkcji usuń instrukcję TODO i dodaj następujący kod, aby zamknąć panel treści i powrócić do panelu listy.

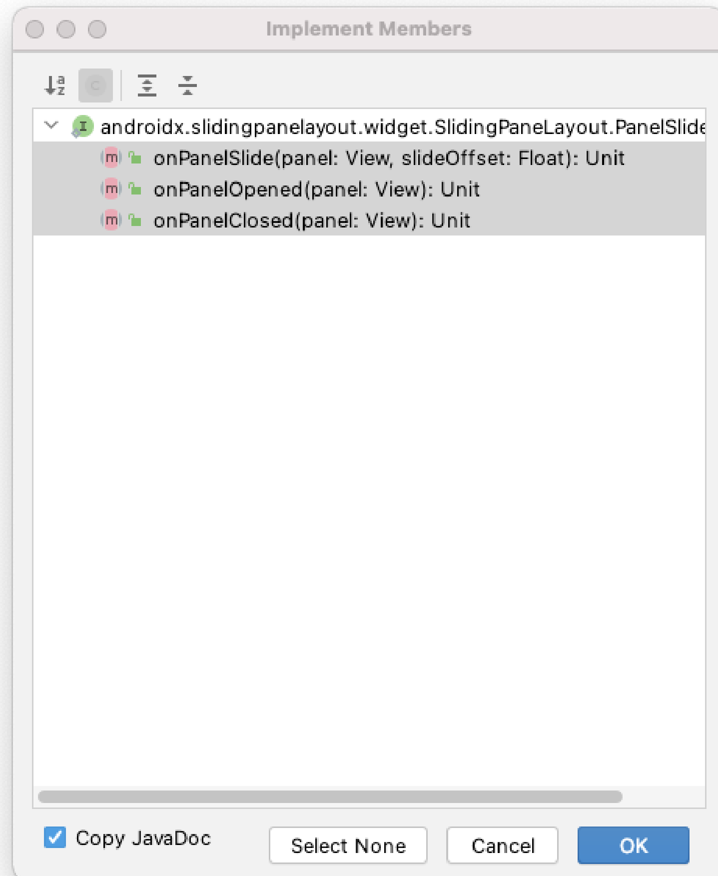
```
slidingPaneLayout.closePane()
```

Uwaga : SlidingPaneLayoutZawsze umożliwia ręczne dzwonienie open()i close()przechodzenie między listą a panelami szczegółów w telefonach. Te metody nie przynoszą efektu, jeśli oba panele są widoczne i nie nakładają się na siebie.

Monitoruj zdarzenia SlidingPaneLayout

Oprócz obsługi zdarzeń prasy wstecznej należy nasłuchiwać i monitorować zdarzenia związane z okienkiem przesuwnym. W miarę przesuwania się okienka zawartości wywołanie zwrotne powinno być odpowiednio włączane lub wyłączane. Wykorzystasz PanelSlideListenerdo tego. Interfejs SlidingPaneLayout.PanelSlideListenerzawiera trzy metody abstrakcyjne onPanelSlide(), onPanelOpened()i onPanelClosed(). Te metody są wywoływane podczas przesuwania, otwierania i zamykania okienka szczegółów.

1. Rozszerz SportsListOnBackPressedCallbackklasę z SlidingPaneLayout.PanelSlideListener.
2. Aby rozwiązać ten błąd, zastosuj trzy metody. Kliknij czerwoną żarówkę i wybierz **Zaimplementuj członków** w Android Studio.



3. Twoja `SportsListOnBackPressedCallback` klasa powinna wyglądać podobnie do następującego:

```
class SportsListOnBackPressedCallback(
  private val slidingPaneLayout: SlidingPaneLayout
): OnBackPressedCallback(slidingPaneLayout.isSlideable && slidingPaneLayout.isOpen),
 SlidingPaneLayout.PanelSlideListener{

  override fun handleOnBackPressed() {
    slidingPaneLayout.closePane()
  }

  override fun onPanelSlide(panel: View, slideOffset: Float) {
    TODO("Not yet implemented")
  }

  override fun onPanelOpened(panel: View) {
    TODO("Not yet implemented")
  }

  override fun onPanelClosed(panel: View) {
    TODO("Not yet implemented")
  }
```

```
}
```

4. Usuń instrukcje TODO.
5. Włącz OnBackPressedCallbackwywołanie zwrotne, gdy okienko szczegółów jest otwarte (jest widoczne) . Można to osiągnąć, wywołując [setEnabled()]funkcję function i przekazując true. Wpisz w środku następujący kod onPanelOpened():

   ```
   setEnabled(true)
   ```

6. Powyższy kod można uprościć za pomocą składni dostępu do właściwości.

   ```
   override fun onPanelOpened(panel: View) {
     isEnabled = true
   }
   ```

7. Podobnie ustaw *isEnabled*na false, gdy okienko szczegółów jest zamknięte.

   ```
   override fun onPanelClosed(panel: View) {
     isEnabled = false
   }
   ```

8. Ostatnim krokiem w wykonaniu wywołania zwrotnego jest dodanie SportsListOnBackPressedCallbackklasy detektora do listy detektorów, które będą powiadamiane o zdarzeniach slajdów w okienku szczegółów. Dodaj initblok do SportsListOnBackPressedCallbackklasy. Wewnątrz initbloku zadzwoń do slidingPaneLayout.addPanelSlideListener()podania this.

   ```
   init {
     slidingPaneLayout.addPanelSlideListener(this)
   }
   ```

   Ukończona SportsListOnBackPressedCallbackklasa powinna wyglądać podobnie do poniższego:

   ```
   class SportsListOnBackPressedCallback(
     private val slidingPaneLayout: SlidingPaneLayout
   ): OnBackPressedCallback(slidingPaneLayout.isSlideable && slidingPaneLayout.isOpen),
    SlidingPaneLayout.PanelSlideListener{

     init {
        slidingPaneLayout.addPanelSlideListener(this)
     }

     override fun handleOnBackPressed() {
        slidingPaneLayout.closePane()
     }

     override fun onPanelSlide(panel: View, slideOffset: Float) {
     }
   ```

```
  override fun onPanelOpened(panel: View) {
    isEnabled = true
  }

  override fun onPanelClosed(panel: View) {
    isEnabled = false
  }
}
```

Zarejestruj oddzwonienie

Aby zobaczyć swoje wywołanie zwrotne w akcji, zarejestruj wywołanie zwrotne za pomocą dyspozytora, OnBackPressedDispatcher.
Klasa bazowa for FragmentActivity, pozwala kontrolować zachowanie przycisku Wstecz za pomocą jego OnBackPressedDispatcher. OnBackPressedDispatcherKontroluje sposób wywoływania zdarzeń przycisku Wstecz do jednego lub większej liczby obiektów OnBackPressedCallback.
Dodaj wywołanie zwrotne za pomocą addCallback()metody. Ta metoda wymaga LifecycleOwner. Gwarantuje to, że OnBackPressedCallbackzostanie dodany tylko wtedy, gdy LifecycleOwnerjest Lifecycle.State.STARTED. Działanie lub fragment usuwa również zarejestrowane wywołania zwrotne, gdy skojarzone z nimi LifecycleOwnerzostaną zniszczone, co zapobiega wyciekom pamięci i sprawia, że nadaje się do użycia we fragmentach lub innych właścicielach cyklu życia, które mają krótszy czas życia.
Metoda addCallback()przyjmuje również w instancji klasę wywołania zwrotnego jako drugi parametr. Zarejestrujesz połączenie zwrotne, wykonując następujące czynności:

1. W SportsListFragmentpliku, wewnątrz funkcji onViewCreated(), tuż pod deklaracją zmiennej wiążącej, utwórz instancję dla SlidingPaneLayouti przypisz do binding.slidingPaneLayoutniej wartość.

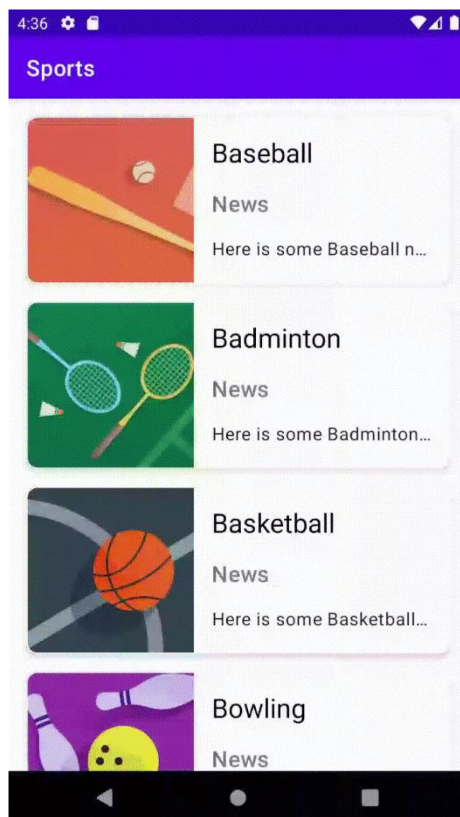
   ```
   val slidingPaneLayout = binding.slidingPaneLayout
   ```

2. W SportsListFragmentpliku, wewnątrz funkcji onViewCreated(), tuż pod deklaracją slidingPaneLayout, dodaj następujący kod:

   ```
   // Connect the SlidingPaneLayout to the system back button.
   requireActivity().onBackPressedDispatcher.addCallback(
     viewLifecycleOwner,
     SportsListOnBackPressedCallback(slidingPaneLayout)
   )
   ```

   Powyższy kod używa addCallback(), przekazując viewLifecycleOwneri instancję SportsListOnBackPressedCallback. To wywołanie zwrotne jest aktywne tylko podczas cyklu życia fragmentu.

3. Czas uruchomić aplikację na emulatorze z profilem telefonu i zobaczyć, jak działa niestandardowy przycisk Wstecz.



10. Tryb blokady

Gdy panele listy i szczegółów nakładają się na mniejsze ekrany, takie jak telefony, użytkownicy mogą domyślnie przesuwać palcem w obu kierunkach, swobodnie przełączając się między dwoma panelami, nawet jeśli nie używają nawigacji gestami . Okienko szczegółów można zablokować lub odblokować, ustawiając tryb blokady SlidingPaneLayout.

1. W emulatorze z profilem telefonu spróbuj przesunąć okienko szczegółów poza ekran.
2. Możesz także przesunąć palcem w okienku szczegółów i spróbować samodzielnie.
3. Nie jest to pożądana funkcja w Twojej aplikacji **Sports .** Dobrym pomysłem jest zablokowanie SlidingPaneLayout, aby uniemożliwić użytkownikom przesuwanie i wysuwanie za pomocą gestów. Aby to zaimplementować, w onViewCreated()metodzie pod slidingPaneLayoutdefinicją ustaw *lockMode*wartość *LOCK_MODE_LOCKED*:

```
slidingPaneLayout.lockMode = SlidingPaneLayout.LOCK_MODE_LOCKED
```

   Aby dowiedzieć się więcej o innych trybach blokady, zapoznaj się z dokumentacją .

   Uwaga : tryb blokady kontroluje tylko to, jakie gesty użytkownika są możliwe. Zawsze możesz programowo otworzyć lub zamknąć **SlidingPaneLayout** , niezależnie od ustawionego trybu blokady.

4. Uruchom aplikację jeszcze raz i zauważ, że okienko szczegółów jest teraz zablokowane.

   Gratulujemy dodania SlidingPaneLayoutdo Twojej aplikacji!

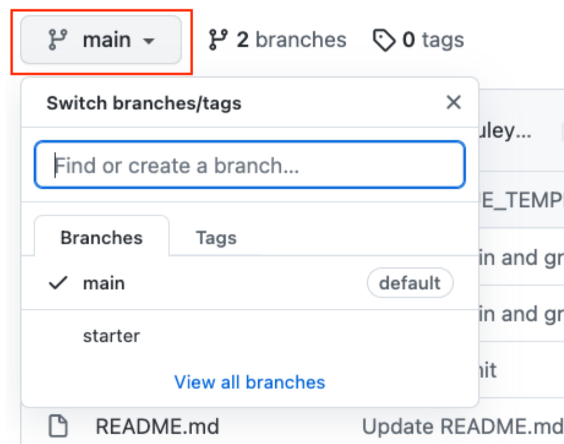
11. Kod rozwiązania

Kod rozwiązania dla tego ćwiczenia programowania znajduje się w projekcie i module pokazanym poniżej.

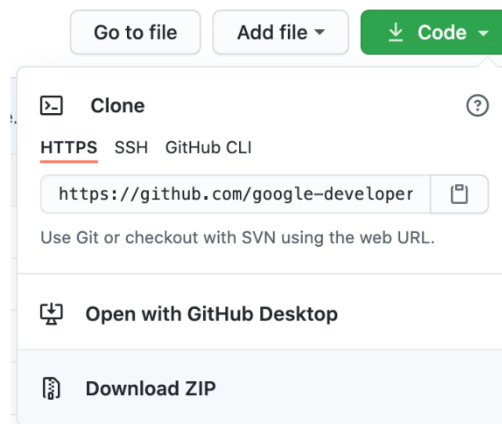
Adres URL kodu rozwiązania: https://github.com/google-developer-training/basic-android-kotlin-training-sports/tree/main

Nazwa oddziału z kodem startowym: main

1. Przejdź do dostarczonej strony repozytorium GitHub dla projektu.
2. Sprawdź, czy nazwa oddziału jest zgodna z nazwą oddziału określoną w ćwiczeniach z programowania. Na przykład na poniższym zrzucie ekranu nazwa gałęzi to **main** .



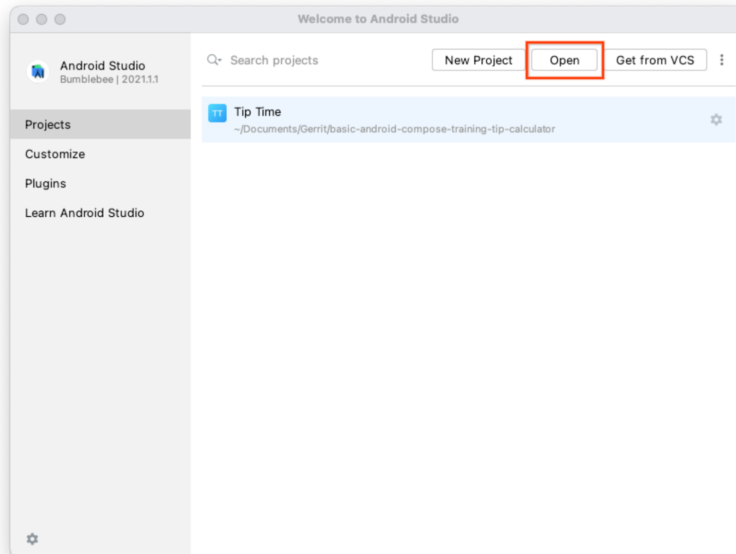
3. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli wyskakujące okienko.



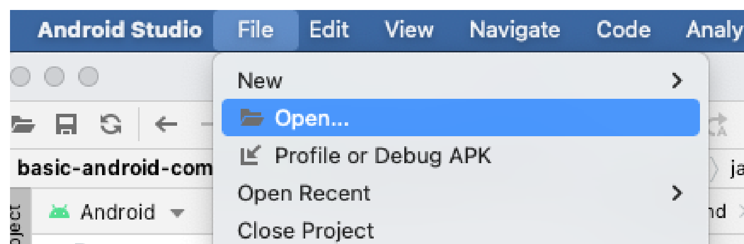
4. W wyskakującym okienku kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na komputerze. Poczekaj na zakończenie pobierania.
5. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
6. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

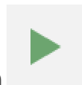
Otwórz projekt w Android Studio

1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz** .



Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Otwórz** .



3. W przeglądarce plików przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.
6. Kliknij przycisk **Uruchom**  , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.

12. Dowiedz się więcej

- Obsługa różnych rozmiarów ekranu
- Dowiedz się o składanych
- Interfejs użytkownika z dużym ekranem w aplikacji Google I/O
- Utwórz układ dwupanelowy
- Widok kontenera fragmentów

Projekt:Aplikacja taca na lunch

1. Zanim zaczniesz

To laboratorium programowania przedstawia nową aplikację o nazwie Lunch Tray, którą zbudujesz samodzielnie. To laboratorium kodowania przeprowadzi Cię przez kolejne kroki, aby ukończyć projekt aplikacji Lunch Tray, w tym konfigurację projektu i testowanie w Android Studio.

To laboratorium programowania różni się od innych w tym kursie. W przeciwieństwie do poprzednich ćwiczeń z programowania, celem tego ćwiczenia z programowania **nie** jest udostępnienie samouczka krok po kroku na temat tworzenia aplikacji. Zamiast tego, to laboratorium ma na celu skonfigurowanie projektu, który wykonasz niezależnie, zapewniając instrukcje, jak ukończyć aplikację i samodzielnie sprawdzić swoją pracę.

Zamiast kodu rozwiązania udostępniamy zestaw testów jako część pobieranej aplikacji. Uruchomisz te testy w Android Studio (pokażemy Ci, jak to zrobić w dalszej części tego ćwiczenia z kodowania) i sprawdzisz, czy Twój kod przejdzie pomyślnie. Może to zająć kilka prób — nawet profesjonalni programiści rzadko przechodzą wszystkie testy przy pierwszej próbie! Gdy Twój kod przejdzie wszystkie testy, możesz uznać ten projekt za ukończony.

Rozumiemy, że możesz po prostu chcieć sprawdzić rozwiązanie. Celowo nie udostępniamy kodu rozwiązania, ponieważ chcemy, abyś przećwiczył, jak to jest być profesjonalnym programistą. Może to wymagać użycia różnych umiejętności, z którymi nie masz jeszcze wiele praktyki, takich jak:

- Terminy Google, komunikaty o błędach i fragmenty kodu w aplikacji, których nie rozpoznajesz;
- Testowanie kodu, odczytywanie błędów, a następnie wprowadzanie zmian w kodzie i ponowne testowanie;
- Powrót do poprzedniej zawartości w Android Basics, aby odświeżyć to, czego się nauczyłeś;
- Porównywanie kodu, o którym wiesz, że działa (tj. kodu podanego w projekcie lub wcześniejszego kodu rozwiązania z innych aplikacji w module 3) z kodem, który piszesz.

Na początku może się to wydawać zniechęcające, ale jesteśmy w 100 procentach pewni, że jeśli udało Ci się ukończyć Część 3, jesteś gotowy na ten projekt. Nie spiesz się i nie poddawaj się. Możesz to zrobić.

Warunki wstępne

- Ten projekt jest przeznaczony dla użytkowników, którzy ukończyli część 3 kursu Android Basics in Kotlin.

Co zbudujesz

- Weźmiesz aplikację do zamawiania jedzenia o nazwie Lunch Tray, zaimplementujesz ViewModel z wiązaniem danych i dodasz nawigację między fragmentami.

Co będziesz potrzebował

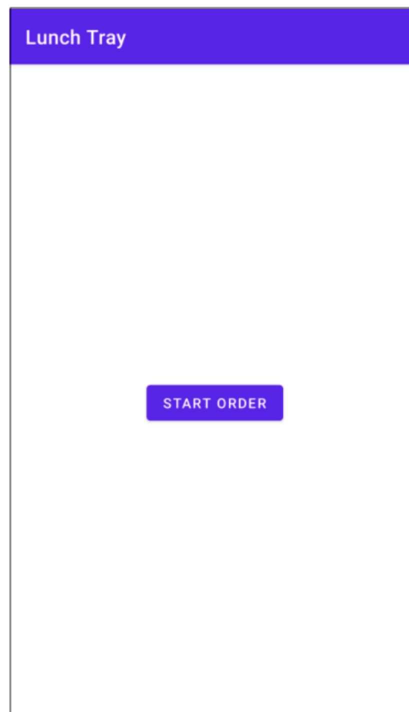
- Komputer z zainstalowanym Android Studio.

2. Zakończony przegląd aplikacji

Witamy w projekcie: taca na lunch!

Jak zapewne wiesz, nawigacja to podstawowa część tworzenia Androida. Niezależnie od tego, czy używasz aplikacji do przeglądania przepisów, znajdowania wskazówek dojazdu do ulubionej restauracji, czy, co najważniejsze, zamawiania jedzenia, prawdopodobnie poruszasz się po wielu ekranach zawartości. W tym projekcie wykorzystasz umiejętności nabyte w części 3, aby zbudować aplikację do zamawiania obiadów o nazwie Lunch Tray, implementując model widoku, powiązanie danych i nawigację między ekranami.

Poniżej znajdują się ostateczne zrzuty ekranu aplikacji. Przy pierwszym uruchomieniu aplikacji Lunch Tray użytkownik jest witany ekranem z jednym przyciskiem z napisem „Rozpocznij zamówienie”.



Po kliknięciu **Rozpocznij zamówienie** , użytkownik może wybrać danie główne z dostępnych opcji. Użytkownik może zmienić swój wybór, co aktualizuje **podsumę** pokazaną na dole.

Lunch Tray

Cauliflower

Whole cauliflower, brined, roasted, and deep fried

$7.00

Three Bean Chili

Black beans, red beans, kidney beans, slow cooked, topped with onion

$4.00

Mushroom Pasta

Penne pasta, mushrooms, basil, with plum tomatoes cooked in garlic and olive oil

$5.50

Spicy Black Bean Skillet

Seasonal vegetables, black beans, house spice blend, served with avocado and quick pickled onions

$5.50

Subtotal: $4.00

CANCEL NEXT

Kolejny ekran pozwala użytkownikowi dodać przystawkę.

Lunch Tray

Summer Salad

Heirloom tomatoes, butter lettuce, peaches, avocado, balsamic dressing

$2.50

Butternut Squash Soup

Roasted butternut squash, roasted peppers, chili oil

$3.00

Spicy Potatoes

Marble potatoes, roasted, and fried in house spice blend

$2.00

Coconut Rice

Rice, coconut milk, lime, and sugar

$1.50

Subtotal: $6.00

CANCEL NEXT

Kolejny ekran pozwala użytkownikowi wybrać akompaniament do zamówienia.

Lunch Tray

Lunch Roll

Fresh baked roll made in house

$0.50

Mixed Berries

Strawberries, blueberries, raspberries, and huckleberries

$1.00

Pickled Veggies

Pickled cucumbers and carrots, made in house

$0.50

Subtotal: $6.50

CANCEL NEXT

Na koniec użytkownikowi wyświetlane jest podsumowanie kosztu zamówienia z podziałem na sumę częściową, podatek od sprzedaży i koszt całkowity. Mogą również złożyć lub anulować zamówienie.

Lunch Tray

ORDER SUMMARY

Three Bean Chili $4.00

Spicy Potatoes $2.00

Pickled Veggies $0.50

Subtotal: $6.50

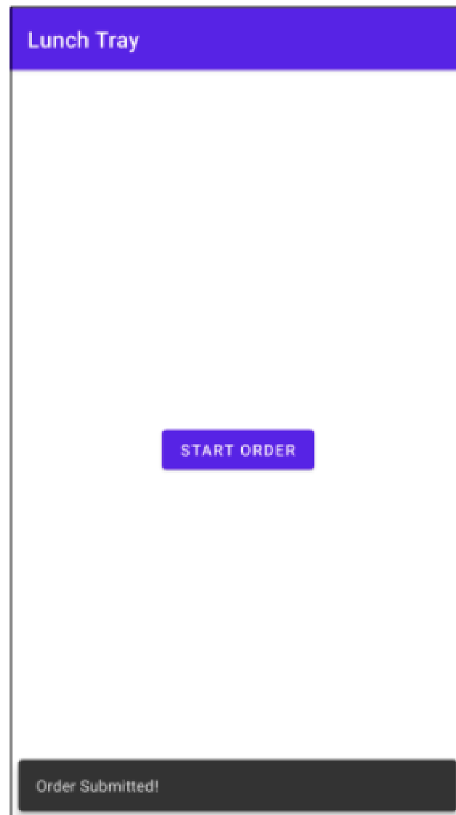
Tax: $0.52

TOTAL: $7.02

SUBMIT ORDER

CANCEL

Obie opcje przywracają użytkownika do pierwszego ekranu. Jeśli użytkownik złożył zamówienie, u dołu ekranu powinien pojawić się toast, informujący o złożeniu zamówienia.



3. Rozpocznij

Pobierz kod projektu

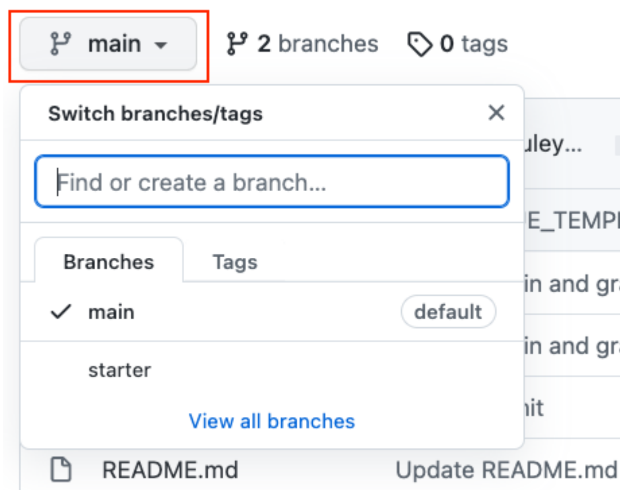
Zauważ, że nazwa folderu to `android-basics-kotlin-lunch-tray-app`. Wybierz ten folder podczas otwierania projektu w Android Studio.

Adres URL kodu startowego:

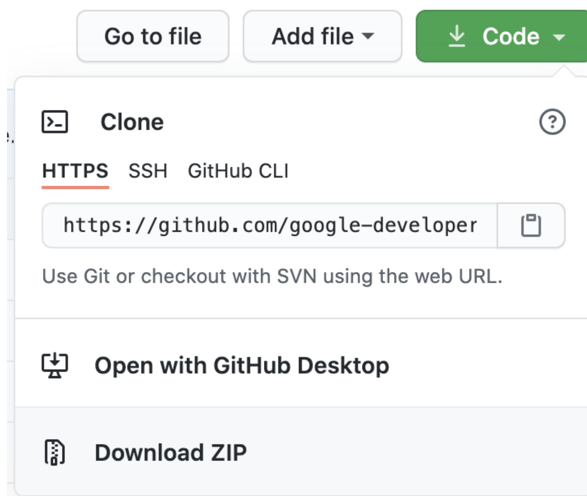
https://github.com/google-developer-training/android-basics-kotlin-lunch-tray-app/tree/main

Nazwa oddziału z kodem startowym:`main`

1. Przejdź do dostarczonej strony repozytorium GitHub dla projektu.
2. Sprawdź, czy nazwa oddziału jest zgodna z nazwą oddziału określoną w ćwiczeniach z programowania. Na przykład na poniższym zrzucie ekranu nazwa gałęzi to **main** .



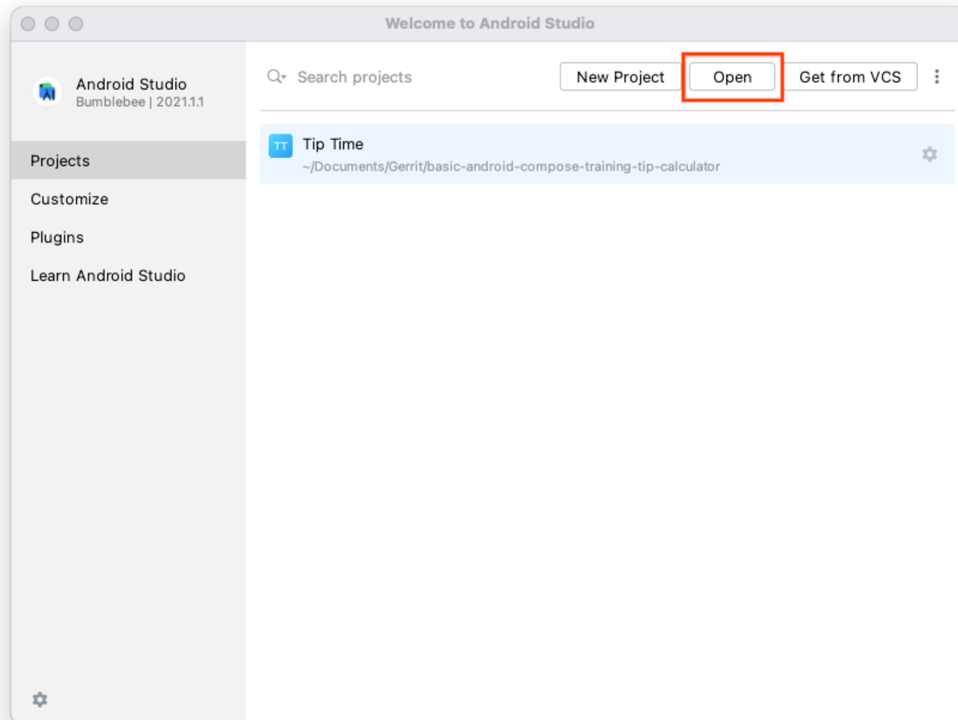
3. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli wyskakujące okienko.



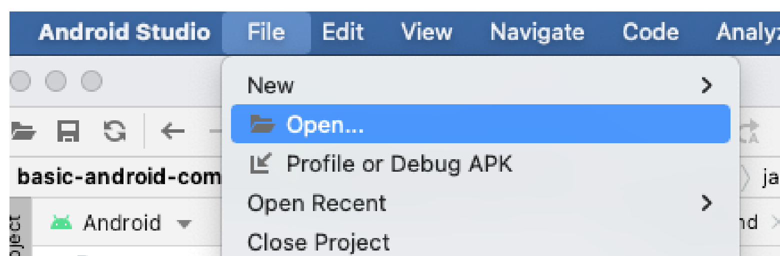
4. W wyskakującym okienku kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na komputerze. Poczekaj na zakończenie pobierania.
5. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
6. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego pliki projektu.

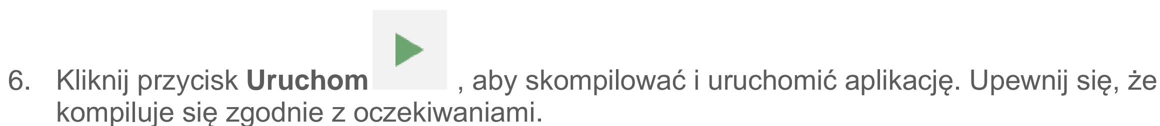
Otwórz projekt w Android Studio

1. Uruchom Android Studio.
2. W oknie **Witamy w Android Studio** kliknij **Otwórz** .



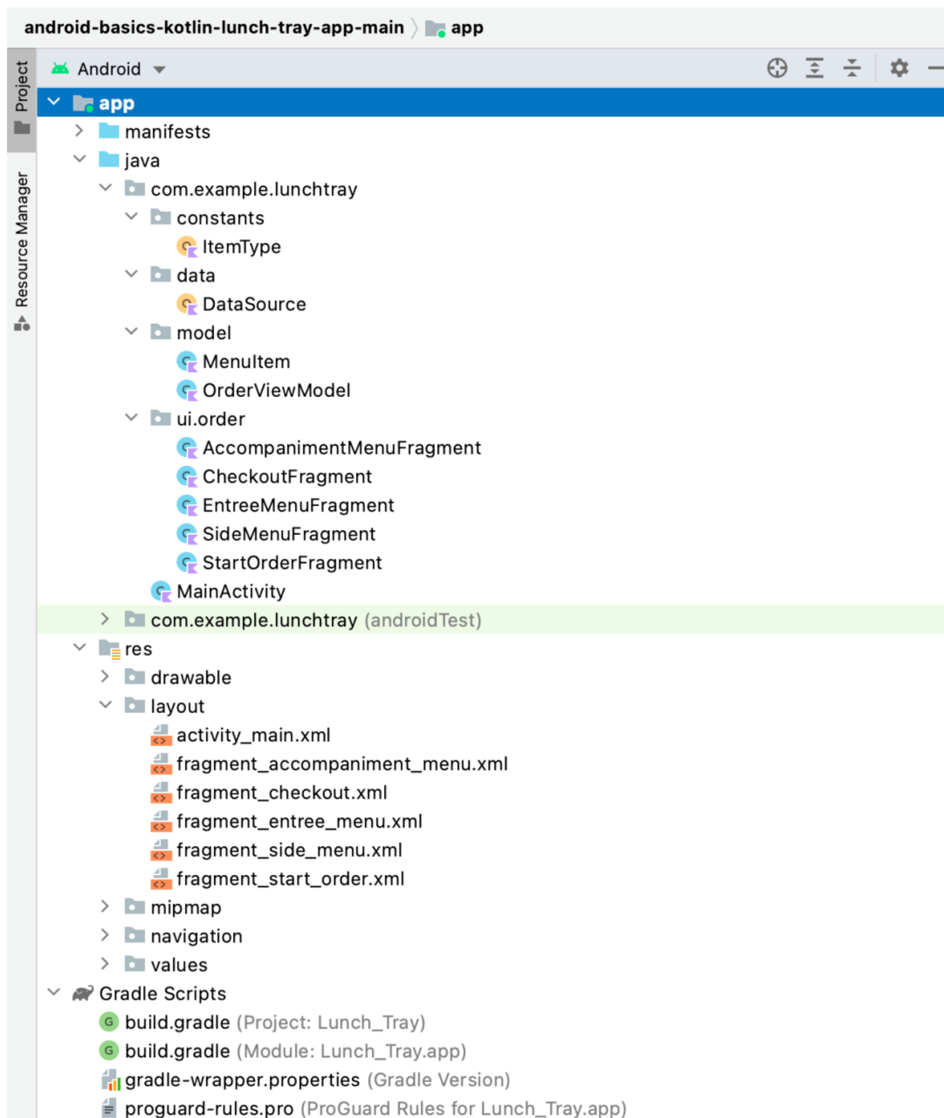
Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Otwórz** .



3. W przeglądarce plików przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
4. Kliknij dwukrotnie ten folder projektu.
5. Poczekaj, aż Android Studio otworzy projekt.
6. Kliknij przycisk **Uruchom**  , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.

   Zanim zaczniesz wdrażanie ViewModeli nawigację, poświęć chwilę, aby upewnić się, że projekt się pomyślnie kompiluje i zapoznaj się z nim. Gdy uruchomisz aplikację po raz pierwszy, zobaczysz pusty ekran. Nie MainActivityprezentuje żadnych fragmentów, ponieważ nie skonfigurowałeś jeszcze wykresu nawigacyjnego.

   Struktura projektu powinna być podobna do innych projektów, nad którymi pracowałeś. Dostępne są oddzielne pakiety dla danych, modelu i interfejsu użytkownika, a także osobne katalogi dla zasobów.



Wszystkie opcje lunchowe, które użytkownik może zamówić (przystawki, przystawki i dodatki) są reprezentowane przez MenuItemklasę w pakiecie **modelowym .** MenuItemobiekty mają nazwę, opis, cenę i typ.

```
data class MenuItem(
    val name: String,
    val description: String,
    val price: Double,
    val type: Int
) {
    fun getFormattedPrice(): String = NumberFormat.getCurrencyInstance().format(price)
}
```

Typ jest reprezentowany przez liczbę całkowitą pochodzącą z ItemTypeobiektu w pakiecie **stałych .**

```
object ItemType {
    val ENTREE = 1
```

```
  val SIDE_DISH = 2
  val ACCOMPANIMENT = 3
}
```

Poszczególne MenuItemobiekty można znaleźć w DataSource.ktpakiecie danych.

```
object DataSource {
  val menuItems = mapOf(
    "cauliflower" to
    MenuItem(
      name = "Cauliflower",
      description = "Whole cauliflower, brined, roasted, and deep fried",
      price = 7.00,
      type = ItemType.ENTREE
    ),
  ...
}
```

Ten obiekt zawiera po prostu mapę składającą się z klucza i odpowiedniego MenuItem. Uzyskasz dostęp do DataSource, ObjectViewModelktóry zaimplementujesz jako pierwszy.

Zdefiniuj ViewModel

Jak widać na zrzutach ekranu na poprzedniej stronie, aplikacja prosi użytkownika o trzy rzeczy: przystawkę, przystawkę i akompaniament. Ekran podsumowania zamówienia wyświetla wtedy sumę częściową i oblicza podatek od sprzedaży na podstawie wybranych pozycji, które są używane do obliczania sumy zamówienia.

Otwórz pakiet **modelu** , OrderViewModel.kta zobaczysz, że kilka zmiennych jest już zdefiniowanych. Właściwość menuItemspo prostu umożliwia dostęp DataSourcedo ViewModel.

```
val menuItems = DataSource.menuItems
```

Po pierwsze, istnieją również zmienne dla previousEntreePrice, previousSidePricei previousAccompanimentPrice. Ponieważ suma częściowa jest aktualizowana, gdy użytkownik dokonuje wyboru (a nie jest sumowana na końcu), zmienne te są używane do śledzenia poprzedniego wyboru użytkownika, jeśli zmieni on swój wybór przed przejściem do następnego ekranu. Wykorzystasz je, aby upewnić się, że sumy częściowe uwzględniają różnicę między cenami poprzednich i aktualnie wybranych produktów.

```
private var previousEntreePrice = 0.0
private var previousSidePrice = 0.0
private var previousAccompanimentPrice = 0.0
```

Istnieją również prywatne zmienne , _entreei _side, _accompanimentsłużące do przechowywania aktualnie wybranego wyboru. Te są typu MutableLiveData<MenuItem?>. Każdemu towarzyszy publiczna właściwość wspierająca , entree, sidei accompaniment, typu niezmiennego LiveData<MenuItem?>. Są one dostępne przez układy fragmentów, aby pokazać

wybrany element na ekranie. Zawarte MenuItemw LiveDataobiekcie również mogą być puste, ponieważ użytkownik może nie wybrać przystawki, strony i/lub akompaniamentu.

```
// Entree for the order
private val _entree = MutableLiveData<MenuItem?>()
val entree: LiveData<MenuItem?> = _entree

// Side for the order
private val _side = MutableLiveData<MenuItem?>()
val side: LiveData<MenuItem?> = _side

// Accompaniment for the order.
private val _accompaniment = MutableLiveData<MenuItem?>()
val accompaniment: LiveData<MenuItem?> = _accompaniment
```

Istnieją również LiveDatazmienne dotyczące sumy częściowej, sumy i podatku, które wykorzystują formatowanie liczb, dzięki czemu są wyświetlane jako waluta.

```
// Subtotal for the order
private val _subtotal = MutableLiveData(0.0)
val subtotal: LiveData<String> = Transformations.map(_subtotal) {
   NumberFormat.getCurrencyInstance().format(it)
}

// Total cost of the order
private val _total = MutableLiveData(0.0)
val total: LiveData<String> = Transformations.map(_total) {
   NumberFormat.getCurrencyInstance().format(it)
}

// Tax for the order
private val _tax = MutableLiveData(0.0)
val tax: LiveData<String> = Transformations.map(_tax) {
   NumberFormat.getCurrencyInstance().format(it)
}
```

Wreszcie, stawka podatku jest zakodowaną wartością 0,08 (8%).

```
private val taxRate = 0.08
```

Istnieje sześć metod OrderViewModel, które musisz wdrożyć.

setEntree(), setSide() i setAccompaniment()

Wszystkie te metody powinny działać w ten sam sposób odpowiednio dla przystawki, boku i akompaniamentu. Jako przykład setEntree()powinien wykonać następujące czynności:

1. Jeśli _entreenie jest null(tj. użytkownik już wybrał danie, ale zmienił swój wybór), ustaw cenę previousEntreePricena .current _entree
2. Jeśli _subtotalnie jest null, odejmij previousEntreePriceod sumy częściowej.
3. Zaktualizuj wartość _entreedo wpisu przekazanego do funkcji (dostęp za MenuItempomocą menuItems).
4. Zadzwoń updateSubtotal(), podając cenę nowo wybranej przystawki.

Logika dla setSide()i setAccompaniment()jest identyczna z implementacją dla setEntree().

aktualizacjaSuma częściowa ()

updateSubtotal()jest wywoływana z argumentem przemawiającym za nową ceną, która powinna zostać dodana do sumy częściowej. Ta metoda powinna zrobić trzy rzeczy:

1. Jeśli _subtotalnie null, dodaj itemPricedo _subtotal.
2. W przeciwnym razie, jeśli _subtotaljest null, ustaw _subtotalna itemPrice.
3. Po _subtotalustawieniu (lub zaktualizowaniu) wywołaj calculateTaxAndTotal(), aby te wartości zostały zaktualizowane w celu odzwierciedlenia nowej sumy częściowej.

oblicz podatek i sumę()

calculateTaxAndTotal()powinien zaktualizować zmienne dotyczące podatku i sumy na podstawie sumy częściowej. Zaimplementuj metodę w następujący sposób:

1. Ustaw _taxrówną stawce podatku pomnożonej przez sumę częściową.
2. Ustaw _totalrówną sumie częściowej plus podatek.

resetOrder()

resetOrder()zostanie wywołany, gdy użytkownik złoży lub anuluje zamówienie. Chcesz się upewnić, że w Twojej aplikacji nie pozostały żadne dane, gdy użytkownik rozpoczyna nowe zamówienie.

Zaimplementuj resetOrder(), ustawiając wszystkie zmodyfikowane zmienne z OrderViewModelpowrotem na ich pierwotną wartość (albo 0.0 lub null).

Utwórz zmienne wiążące dane

Zaimplementuj powiązanie danych w plikach układu. Otwórz pliki układu i dodaj zmienne wiązania danych typu OrderViewModeli/lub odpowiedniej klasy fragmentu.

Musisz zaimplementować wszystkie TODOkomentarze, aby ustawić detektory tekstu i kliknięć w czterech plikach układu:

1. fragment_entree_menu.xml
2. fragment_side_menu.xml
3. fragment_accompaniment_menu.xml
4. fragment_checkout.xml

Każde konkretne zadanie jest odnotowane w komentarzu TODO w plikach układu, ale kroki są podsumowane poniżej.

1. W tagu dodaj fragment_entree_menu.xmlzmienną <data>powiązania dla EntreeMenuFragment. Dla każdego z przycisków opcji musisz ustawić przystawkę w miejscu ViewModelzaznaczenia. Tekst

widoku sumy częściowej tekstu powinien zostać odpowiednio zaktualizowany. Musisz także ustawić onClickatrybut dla cancel_buttoni next_button, aby anulować zamówienie lub odpowiednio przejść do następnego ekranu.

2. Zrób to samo w fragment_side_menu.xml, dodając zmienną powiązania dla SideMenuFragment, z wyjątkiem ustawienia strony w modelu widoku, gdy każdy przycisk opcji jest zaznaczony. Tekst sumy częściowej również będzie musiał zostać zaktualizowany, a także trzeba będzie ustawić onClickatrybut dla przycisków anuluj i dalej.
3. Zrób to samo jeszcze raz, ale w fragment_accompaniment_menu.xml, tym razem ze zmienną powiązania dla AccompanimentMenuFragment, ustawiając akompaniament po wybraniu każdego przycisku opcji. Ponownie, musisz również ustawić atrybuty dla tekstu sumy częściowej, przycisk Anuluj i przycisk Dalej.
4. W fragment_checkout.xmlprogramie musisz dodać <data>tag, aby móc zdefiniować zmienne wiążące. W <data>tagu dodaj dwie zmienne wiążące, jedną dla OrderViewModel, a drugą dla CheckoutFragment. W widokach tekstowych musisz ustawić nazwy i ceny wybranego przystawki, przystawki i akompaniamentu z OrderViewModel. Musisz także ustawić sumę częściową, podatek i sumę z OrderViewModel. Następnie, onClickAttributeskorzystając z odpowiednich funkcji z CheckoutFragment.

.

Wskazówka: Jeśli potrzebujesz odświeżenia, możesz przejrzeć Dodaj zmienne powiązania danych i Użyj wyrażeń powiązania w Użyj LiveData z ViewModel.

Zainicjuj zmienne wiążące dane we fragmentach

Zainicjuj zmienne wiązania danych w odpowiednich plikach fragmentów wewnątrz metody, onViewCreated().

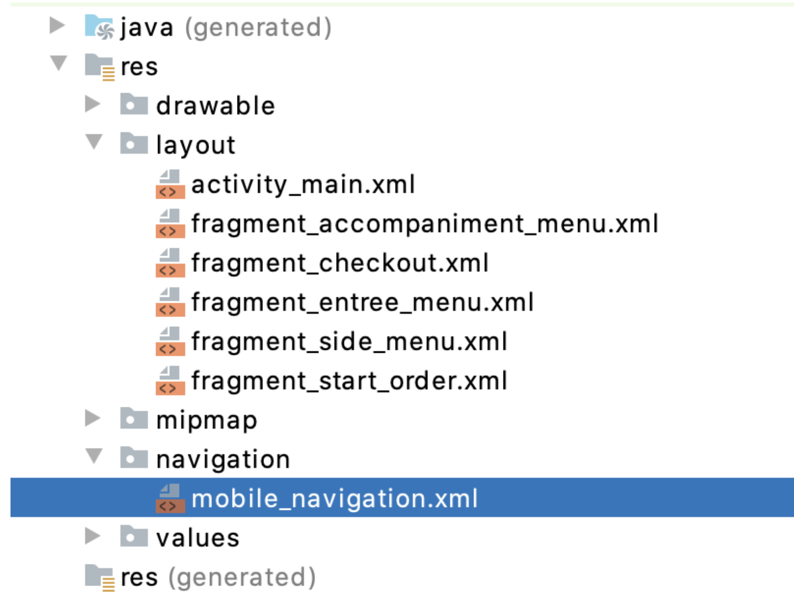
1. EntreeMenuFragment
2. SideMenuFragment
3. AccompanimentMenuFragment
4. CheckoutFragment

Utwórz wykres nawigacyjny

Jak dowiedziałeś się w części 3, wykres nawigacyjny jest przechowywany w FragmentContainerView, zawartym w działaniu. Otwórz activity_main.xmli zastąp TODO następującym kodem, aby zadeklarować FragmentContainerView.

```
<androidx.fragment.app.FragmentContainerView
   android:id="@+id/nav_host_fragment"
   android:name="androidx.navigation.fragment.NavHostFragment"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   app:defaultNavHost="true"
   app:navGraph="@navigation/mobile_navigation"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintLeft_toLeftOf="parent"
   app:layout_constraintRight_toRightOf="parent"
   app:layout_constraintTop_toTopOf="parent" />
```

Wykres nawigacji mobile_navigation.xmlznajduje się w pakiecie **res.navigation** .



To jest wykres nawigacyjny dla aplikacji. Jednak plik jest obecnie pusty. Twoim zadaniem jest dodanie miejsc docelowych do wykresu nawigacji i zamodelowanie następującej nawigacji między ekranami.

1. Nawigacja z StartOrderFragmentdoEntreeMenuFragment
2. Nawigacja z EntreeMenuFragmentdoSideMenuFragment
3. Nawigacja z SideMenuFragmentdoAccompanimentMenuFragment
4. Nawigacja z AccompanimentMenuFragmentdoCheckoutFragment
5. Nawigacja z CheckoutFragmentdoStartOrderFragment
6. Nawigacja z EntreeMenuFragmentdoStartOrderFragment
7. Nawigacja z SideMenuFragmentdoStartOrderFragment
8. Nawigacja z AccompanimentMenuFragmentdoStartOrderFragment
9. Miejscem **docelowym początkowym** powinno byćStartOrderFragment

Po skonfigurowaniu wykresu nawigacji musisz przeprowadzić nawigację w klasach fragmentów. Pozostałe TODOkomentarze zaimplementuj we fragmentach, a także MainActivity.kt.

1. W przypadku goToNextScreen()metody w EntreeMenuFragment, SideMenuFragmenti AccompanimentMenuFragmentprzejdź do następnego ekranu w aplikacji.
2. W przypadku cancelOrder()metody w EntreeMenuFragment, SideMenuFragment, AccompanimentMenuFragmenti CheckoutFragment, najpierw wywołaj resetOrder(), sharedViewModela następnie przejdź do StartOrderFragment.
3. W StartOrderFragmentprogramie zaimplementuj , setOnClickListener()aby przejść do EntreeMenuFragment.
4. W CheckoutFragmentprogramie zaimplementuj submitOrder()metodę. Zadzwoń resetOrder()na sharedViewModel, a następnie przejdź do StartOrderFragment.
5. Na koniec w MainActivity.kt, ustaw navControllerna navControllerz NavHostFragment.

Uwaga: wszystkie dane tej aplikacji są obsługiwane przez OrderViewModel. Nie będziesz musiał przekazywać żadnych argumentów do fragmentów docelowych.

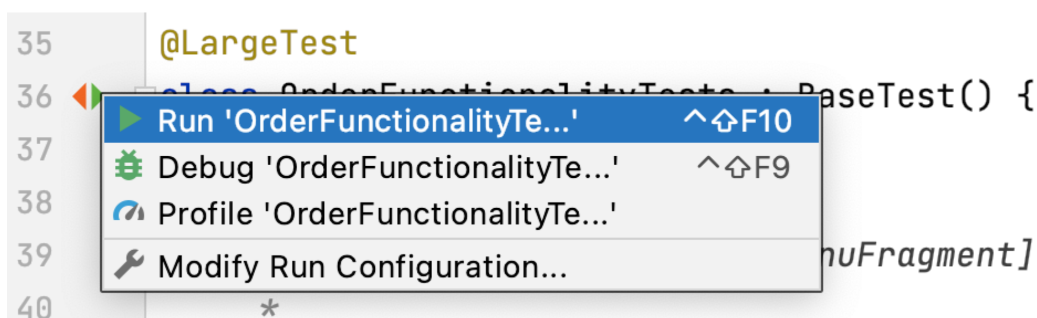
4. Przetestuj swoją aplikację

Projekt Lunch Tray zawiera obiekt docelowy „androidTest" z kilkoma przypadkami testowymi: MenuContentTests, NavigationTestsi OrderFunctionalityTests.

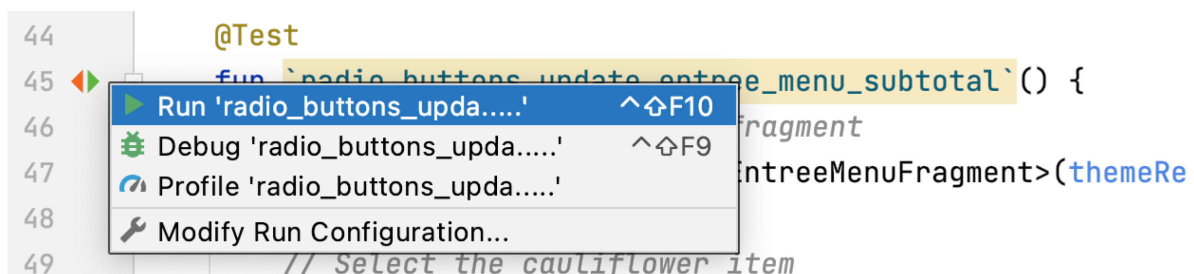
Przeprowadzanie testów

Aby uruchomić testy, możesz wykonać jedną z następujących czynności:

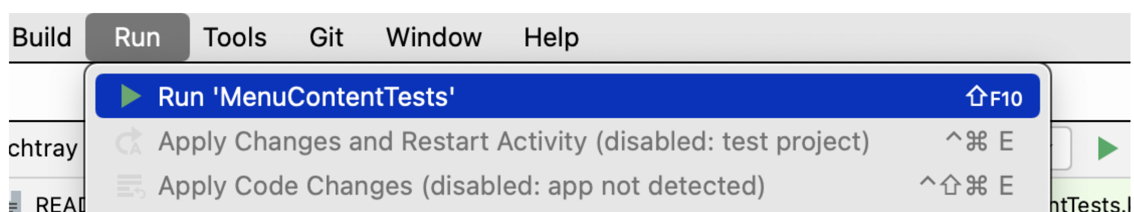
W przypadku pojedynczego przypadku testowego otwórz klasę przypadku testowego i kliknij zieloną strzałkę po lewej stronie deklaracji klasy. Następnie możesz wybrać z menu opcję Uruchom. Spowoduje to uruchomienie wszystkich testów w przypadku testowym.



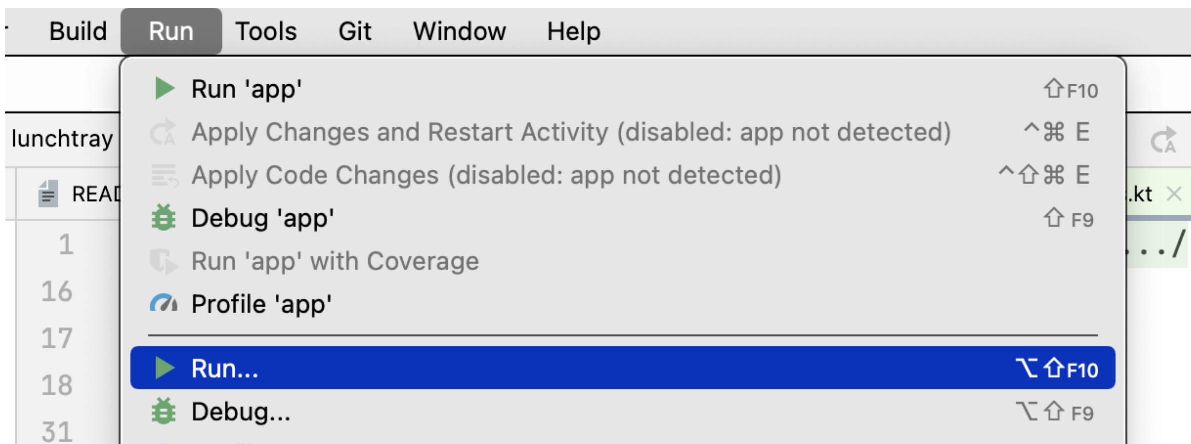
Często będziesz chciał uruchomić tylko jeden test, na przykład, jeśli tylko jeden test się nie powiedzie, a inne testy zakończą się pomyślnie. Pojedynczy test można uruchomić tak samo, jak cały przypadek testowy. Użyj zielonej strzałki i wybierz opcję **Uruchom** .



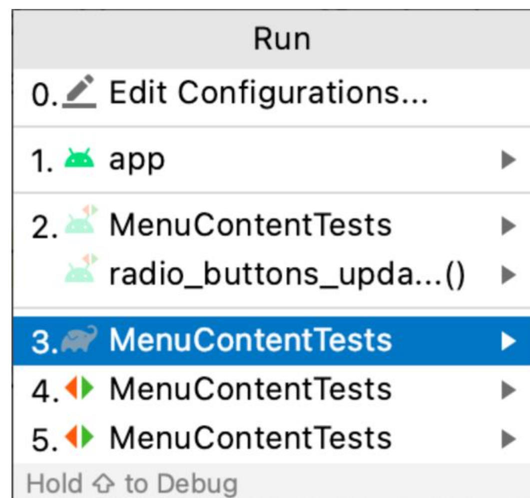
Jeśli masz wiele przypadków testowych, możesz również uruchomić cały zestaw testów. Podobnie jak w przypadku uruchamiania aplikacji, tę opcję znajdziesz w menu **Uruchom** .



Zwróć uwagę, że Android Studio domyślnie dopasuje się do ostatniego uruchomionego celu (aplikacji, celów testowych itp.), więc jeśli w menu nadal pojawi się komunikat **Uruchom > Uruchom „aplikację”** , możesz uruchomić cel testu, wybierając **Uruchom > Uruchom** .



Następnie wybierz cel testowy z menu podręcznego.



5. Opcjonalnie: przekaż nam swoją opinię!

Chętnie poznamy Twoje opinie na temat tego projektu. Wypełnij tę krótką ankietę, aby przekazać nam swoją opinię — Twoja opinia pomoże w kierowaniu przyszłymi projektami w ramach tego kursu.