Część 2: Układy

Ulepsz interfejs użytkownika swojej aplikacji, poznając układy, wytyczne dotyczące projektowania materiałów i najlepsze praktyki dotyczące tworzenia interfejsu użytkownika.

Spis treści

Ścieżka 1. Uzyskaj dane wejściowe użytkownika w aplikacji: część 1

Utwórz aplikację kalkulatora napiwków, najpierw tworząc układ, a następnie wdrażając logikę, aby obliczyć napiwek na podstawie danych wejściowych użytkownika.

Klasy i dziedziczenie w Kotlinie

1. Zanim zaczniesz

Warunki wstępne

- Znajomość korzystania z Kotlin Playground do edycji programów Kotlin.
- Podstawowe pojęcia z zakresu programowania w Kotlinie nauczane w części 1 tego kursu. W szczególności `main()`program działa z argumentami zwracającymi wartości, zmienne, typy danych i operacje oraz `if/else`instrukcje.
- Potrafi zdefiniować klasę w Kotlinie, utworzyć z niej instancję obiektu i uzyskać dostęp do jej właściwości i metod.

Czego się nauczysz

- Utwórz program Kotlin, który wykorzystuje dziedziczenie do implementacji hierarchii klas.
- Rozszerz klasę, zastąp jej istniejącą funkcjonalność i dodaj nową funkcjonalność.
- Wybierz odpowiedni modyfikator widoczności dla zmiennych.

Co zbudujesz

- Program Kotlin z różnymi typami mieszkań, które są realizowane w hierarchii klasowej.

Czego potrzebujesz

- Komputer z połączeniem internetowym umożliwiającym dostęp do Placu Zabaw Kotlin

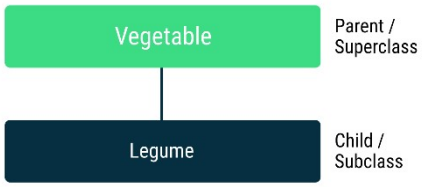
2. Czym jest hierarchia klas?

Jest rzeczą naturalną, że ludzie klasyfikują przedmioty, które mają podobne właściwości i zachowanie do grup, a nawet tworzą między nimi pewien rodzaj hierarchii. Na przykład możesz mieć szeroką kategorię, taką jak warzywa, a w jej ramach możesz mieć bardziej szczegółowy rodzaj, taki jak rośliny strączkowe . W roślinach strączkowych można na przykład mieć jeszcze bardziej specyficzne rodzaje, takie jak groch, fasola, soczewica, ciecierzyca i soja.

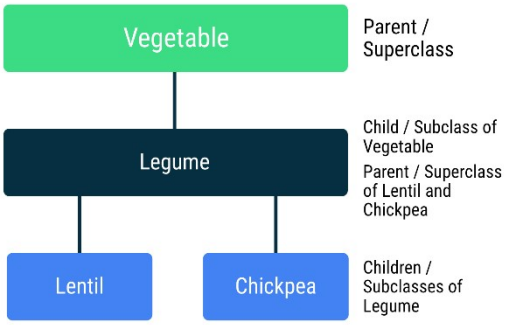
Można to przedstawić hierarchicznie, ponieważ rośliny strączkowe zawierają lub *dziedziczą* wszystkie właściwości warzyw (np. są roślinami i są jadalne). Podobnie groch, fasola i soczewica mają właściwości roślin strączkowych plus swoje unikalne właściwości.

Przyjrzyjmy się, jak mógłbyś przedstawić tę relację w kategoriach programistycznych. Jeśli tworzysz `Vegetable`klasę w Kotlinie, możesz utworzyć `Legume`jako *dziecko* lub *podklasę* klasy `Vegetable`. Oznacza to, że wszystkie właściwości i metody `Vegetable`klasy są dziedziczone (co oznacza również dostępne w) `Legume`klasie.

Możesz to przedstawić na diagramie *hierarchii klas,* jak pokazano poniżej. Możesz odnosić się do `Vegetable`klasy *nadrzędnej* lub *nadrzędnej*`Legume` .



Możesz kontynuować i rozszerzać hierarchię klas, tworząc podklasy, `Legume`takie jak `Lentil`i `Chickpea`. Czyni to `Legume`zarówno klasę potomną lub podklasę, `Vegetable`jak i klasę nadrzędną lub nadrzędną `Lentil`i `Chickpea`. `Vegetable`jest klasą *główną* lub * najwyższego poziomu (*lub *podstawową)* tej hierarchii.



Uwaga: Podsumowanie terminologii

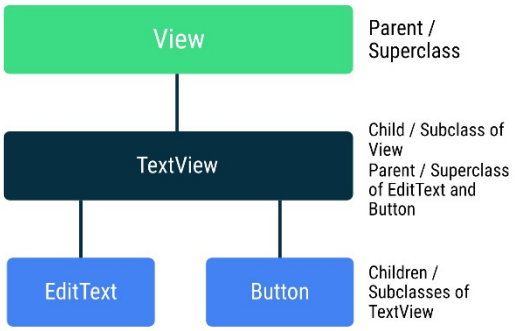
Oto podsumowanie słów użytych w tym laboratorium i ich znaczenia w kontekście klas Kotlin.

- **Hierarchia klas.** Układ, w którym zajęcia są zorganizowane w hierarchii rodziców i dzieci. Diagramy hierarchii są zwykle rysowane z rodzicami pokazanymi nad dziećmi.
- **Dziecko lub podklasa.** Każda klasa, która znajduje się poniżej innej klasy w hierarchii.
- **Klasa nadrzędna, nadklasa lub klasa podstawowa.** Dowolna klasa z co najmniej jedną klasą podrzędną.
- **Klasa główna lub klasa najwyższego poziomu.** Klasa na szczycie (lub korzeniu) hierarchii klas.
- **Dziedzictwo.** Kiedy klasa potomna zawiera (lub dziedziczy) wszystkie właściwości i metody swojej klasy nadrzędnej. Pozwala to na udostępnianie i ponowne wykorzystywanie kodu, dzięki czemu programy są łatwiejsze do zrozumienia i utrzymania.

Dziedziczenie w klasach Androida

Chociaż możesz pisać kod Kotlin bez używania klas, co robiłeś w poprzednich ćwiczeniach z programowania, wiele części Androida jest dostarczanych w postaci klas, w tym działań, widoków i grup widoków. Zrozumienie hierarchii klas ma zatem fundamentalne znaczenie dla tworzenia aplikacji na Androida i umożliwia korzystanie z funkcji udostępnianych przez platformę Android.

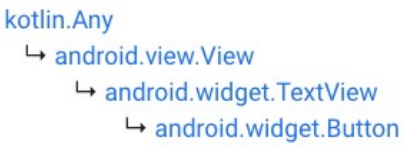
Na przykład w systemie Android istnieje Viewklasa, która reprezentuje prostokątny obszar na ekranie i jest odpowiedzialna za rysowanie i obsługę zdarzeń. Klasa TextViewjest podklasą Viewklasy, co oznacza, że TextViewdziedziczy wszystkie właściwości i funkcje z Viewklasy, plus dodaje specyficzną logikę wyświetlania tekstu użytkownikowi.



Idąc o krok dalej, klasy EditTexti Buttonsą dziećmi TextViewklasy. Dziedziczą wszystkie właściwości i metody klas TextViewi Vieworaz dodają własną, specyficzną logikę. Na przykład EditTextdodaje własną funkcjonalność umożliwiającą edycję tekstu na ekranie.

Zamiast kopiować i wklejać całą logikę z klas Viewi TextViewdo EditTextklasy, EditTextmożna po prostu podklasować TextViewklasę, która z kolei podklasuje Viewklasę. Następnie kod w EditTextklasie może skupić się konkretnie na tym, co odróżnia ten komponent interfejsu użytkownika od innych widoków.

U góry strony dokumentacji dla klasy Androida w witrynie internetowej developer.android.com można zobaczyć diagram hierarchii klas. Jeśli widzisz kotlin.Anyna szczycie hierarchii, to dlatego, że w Kotlinie wszystkie klasy mają wspólną nadklasę Any. Dowiedz się więcej tutaj .



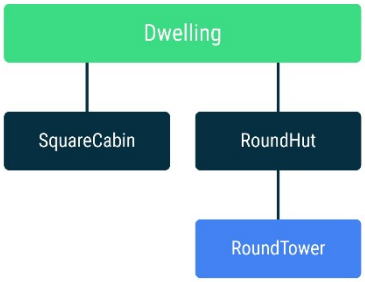
Jak widać, nauka wykorzystywania dziedziczenia między klasami może ułatwić pisanie, ponowne używanie, odczytywanie i testowanie kodu.

3. Utwórz klasę bazową

Hierarchia klasowa mieszkań

W tym laboratorium z programowania zbudujesz program Kotlin, który zademonstruje, jak działają hierarchie klas, na przykładzie mieszkań (schronisk, w których mieszkają ludzie) z powierzchnią podłogi, historiami i mieszkańcami.

Poniżej znajduje się diagram hierarchii klas, którą zamierzasz zbudować. U podstaw znajduje się a Dwelling, które określa właściwości i funkcjonalność, które są prawdziwe dla wszystkich mieszkań, podobnie jak plan. Następnie masz zajęcia dla kwadratowej kabiny (SquareCabin), okrągłej chaty (RoundHut) i okrągłej wieży (RoundTower), która jest RoundHutwielopiętrowa.



Klasy, które zaimplementujesz:

- `Dwelling`: klasa bazowa reprezentująca niespecyficzne schronienie, które zawiera informacje wspólne dla wszystkich mieszkań.
- `SquareCabin`: kwadratowa kabina wykonana z drewna o kwadratowej powierzchni podłogi.
- `RoundHut`: okrągła chata zrobiona ze słomy z okrągłą podłogą i rodzicem `RoundTower`.
- `RoundTower`: okrągła wieża z kamienia z okrągłą podłogą i wieloma kondygnacjami.

Utwórz abstrakcyjną klasę mieszkania

Każda klasa może być klasą bazową hierarchii klas lub rodzicem innych klas.

Klasa „abstrakcyjna" to klasa, której nie można utworzyć, ponieważ nie jest w pełni zaimplementowana. Możesz myśleć o tym jako o szkicu. Szkic zawiera pomysły i plany na coś, ale zwykle nie ma wystarczającej ilości informacji, aby to zbudować. Używasz szkicu (klasy abstrakcyjnej), aby stworzyć plan (klasę), z którego zbudujesz rzeczywistą instancję obiektu.

Powszechną korzyścią tworzenia nadklasy jest zawieranie właściwości i funkcji, które są wspólne dla wszystkich jej podklas. Jeśli wartości właściwości i implementacji funkcji nie są znane, uczyń klasę abstrakcyjną. Na przykład `Vegetables`mieć wiele właściwości wspólnych dla wszystkich warzyw, ale nie możesz utworzyć wystąpienia niespecyficznego warzywa, ponieważ nie znasz na przykład jego kształtu lub koloru. Podobnie `Vegetable`jest z klasą abstrakcyjną, która pozostawia podklasom określenie konkretnych szczegółów dotyczących każdego warzywa.

Deklaracja klasy abstrakcyjnej zaczyna się od `abstract`słowa kluczowego.

Uwaga: natkniesz się na wiele abstrakcyjnych klas w Androidzie, które będziesz musiał podklasy w przyszłych ćwiczeniach z programowania w tym kursie.

`Dwelling`będzie klasą abstrakcyjną, taką jak `Vegetable`. Będzie zawierać właściwości i funkcje wspólne dla wielu typów mieszkań, ale dokładne wartości właściwości i szczegóły realizacji funkcji nie są znane.

1. Przejdź do placu zabaw Kotlin pod adresem https://developer.android.com/training/kotlinplayground .
2. W edytorze usuń `println("Hello, world!")`wewnątrz `main()`funkcji.
3. Następnie dodaj ten kod poniżej `main()`funkcji, aby utworzyć `abstract`klasę o nazwie `Dwelling`.

```
abstractclassDwelling(){
}
```

Dodaj nieruchomość dla materiałów budowlanych

W tej `Dwelling`klasie definiujesz rzeczy, które są prawdziwe dla wszystkich mieszkań, nawet jeśli mogą być różne dla różnych mieszkań. Wszystkie mieszkania wykonane są z jakiegoś materiału budowlanego.

1. Wewnątrz Dwellingutwórz buildingMaterialzmienną typu String, która będzie reprezentować materiał budowlany. Ponieważ materiał budowlany się nie zmieni, użyj , valaby uczynić go zmienną niezmienną.

```
valbuildingMaterial:String
```

1. Uruchom swój program i pojawi się ten błąd.

```
Propertymust be initializedor be abstract
```

Właściwość buildingMaterialnie ma wartości. W rzeczywistości NIE MOŻESZ nadać jej wartości, ponieważ niespecyficzny budynek nie jest zrobiony z niczego konkretnego. Tak więc, jak wskazuje komunikat o błędzie, deklarację można poprzedzić buildingMaterialsłowem abstractkluczowym, aby wskazać, że nie będzie ono tutaj definiowane.

1. Dodaj abstractsłowo kluczowe do definicji zmiennej.

```
abstractvalbuildingMaterial:String
```

1. Uruchom swój kod i chociaż nic nie robi, teraz kompiluje się bez błędów.
2. Utwórz wystąpienie Dwellingw main()funkcji i uruchom swój kod.

```
valdwelling=Dwelling()
```

1. Otrzymasz błąd, ponieważ nie możesz utworzyć instancji Dwellingklasy abstrakcyjnej.

```
Cannotcreateaninstance of anabstractclass
```

1. Usuń ten niepoprawny kod.

Twój gotowy kod do tej pory:

```
abstractclassDwelling(){
  abstractvalbuildingMaterial:String
}
```

Dodaj właściwość dla pojemności

Kolejną właściwością mieszkania jest pojemność, czyli ile osób może w nim mieszkać.

Wszystkie mieszkania mają niezmienną pojemność. Jednak pojemności nie można ustawić w ramach Dwellingsuperklasy. Powinien być określony w podklasach dla określonych typów mieszkań.

1. W DwellingDodaj abstractliczbę całkowitą valo nazwie capacity.

```
abstractvalcapacity:Int
```

Dodaj prywatną nieruchomość dla liczby mieszkańców

Wszystkie mieszkania będą miały określoną liczbę osób residentsmieszkających w mieszkaniu (która może być mniejsza lub równa capacity), więc zdefiniuj residentswłaściwość w Dwellingnadklasie, aby wszystkie podklasy miały dziedziczyć i używać.

1. Możesz utworzyć residentsparametr, który jest przekazywany do konstruktora Dwellingklasy. Właściwość residentsto var, ponieważ liczba mieszkańców może się zmienić po utworzeniu instancji.

```
abstractclassDwelling(privatevarresidents:Int){
```

Zauważ, że residentswłaściwość jest oznaczona privatesłowem kluczowym. Private to modyfikator widoczności w Kotlin, co oznacza, że residentswłaściwość jest widoczna tylko dla tej klasy (i może być używana wewnątrz). Nie można uzyskać do niego dostępu z innego miejsca w programie. Możesz oznaczyć właściwości lub metody za pomocą słowa kluczowego private. W przeciwnym razie, gdy nie określono modyfikatora widoczności, właściwości i metody są publicdomyślnie dostępne z innych części programu. Ponieważ liczba osób mieszkających w mieszkaniu to zazwyczaj informacja prywatna (w porównaniu do informacji o materiale budowlanym lub pojemności budynku), jest to rozsądna decyzja.

Po zdefiniowaniu zarówno capacitymieszkania, jak i numeru prądu residents, można utworzyć funkcję hasRoom()określającą, czy w mieszkaniu jest miejsce dla innego mieszkańca. Możesz zdefiniować i zaimplementować hasRoom()funkcję w Dwellingklasie, ponieważ wzór na obliczenie, czy jest miejsce, jest taki sam dla wszystkich mieszkań. W a jest miejsce, Dwellingjeśli liczba residentsjest mniejsza niż capacity, a funkcja powinna zwracać truelub falseopierać się na tym porównaniu.

1. Dodaj hasRoom()funkcję do Dwellingklasy.

```
funhasRoom():Boolean{
  returnresidents<capacity
}
```

1. Możesz uruchomić ten kod i nie powinno być żadnych błędów. Nie robi jeszcze nic widocznego.

Twój wypełniony kod powinien wyglądać tak:

```
abstractclassDwelling(privatevarresidents:Int){

  abstractvalbuildingMaterial:String
  abstractvalcapacity:Int

  funhasRoom():Boolean{
    returnresidents<capacity
  }
}
```

4. Utwórz podklasy

Utwórz podklasę SquareCabin

1. Poniżej Dwellingklasy utwórz klasę o nazwie SquareCabin.

```
classSquareCabin
```

1. Następnie musisz wskazać, że SquareCabinjest to powiązane z Dwelling. W swoim kodzie chcesz wskazać, że SquareCabinextends from Dwelling(lub jest podklasą do Dwelling)ponieważ SquareCabinzapewni implementację dla abstrakcyjnych części Dwelling.

Wskaż tę relację dziedziczenia, dodając dwukropek (:) po nazwie klasy, a następnie wywołaj inicjalizację klasy SquareCabinnadrzędnej . DwellingNie zapomnij dodać nawiasów po Dwellingnazwie klasy.

```
classSquareCabin:Dwelling()
```

1. Wychodząc z superklasy, musisz przekazać wymagane parametry, których oczekuje ta superklasa. Dwellingwymaga liczby residentsjako danych wejściowych. Możesz przekazać ustaloną liczbę mieszkańców, takich jak 3.

```
classSquareCabin:Dwelling(3)
```

Jednak chcesz, aby Twój program był bardziej elastyczny i umożliwiał zmienną liczbę mieszkańców dla SquareCabins. Dlatego utwórz residentsparametr w SquareCabindefinicji klasy. Nie deklaruj residentsjako val,, ponieważ ponownie używasz właściwości już zadeklarowanej w klasie nadrzędnej Dwelling.

```
classSquareCabin(residents:Int):Dwelling(residents)
```

Uwaga: Wiele się dzieje pod maską z tymi definicjami klas.

W nagłówku zajęć widziszclassSquareCabin(residents: Int) …

To jest właściwie skrót odclassSquareCabinconstructor(residents: Int) …

Jest constructorwywoływana podczas tworzenia instancji obiektu z klasy. Na przykład podczas wywołania SquareCabin(4)funkcji constructorof SquareCabinjest wywoływane w celu zainicjowania instancji obiektu.

Buduje twoją instancję na constructorpodstawie wszystkich informacji w klasie, w tym przekazanych argumentów. Kiedy klasa dziedziczy właściwości i funkcje z rodzica, constructorwywołuje constructorklasę rodzica w celu zakończenia inicjowania instancji obiektu.

W związku z tym, gdy tworzysz instancję za pomocą SquareCabin(4), wykonywane jest constructorfor SquareCabin, a ze względu na relację dziedziczenia Dwellingwykonywany jest również konstruktor. Definicja Dwellingklasy określa, że jej konstruktor wymaga residentsparametru, dlatego w definicji klasy widzisz residentsprzekazany do Dwellingkonstruktora SquareCabin. Więcej o konstruktorach dowiesz się w późniejszych ćwiczeniach z programowania.

1. Uruchom swój kod.
2. Spowoduje to błędy. Spójrz:

```
Class 'SquareCabin' is not abstract and does not implementabstractbaseclassmember public abstractvalbuildingMaterial: String defined in Dwelling
```

Kiedy deklarujesz abstrakcyjne funkcje i zmienne, jest to jak obietnica, że później przekażesz im wartości i implementacje. W przypadku zmiennej oznacza to, że każda podklasa tej klasy abstrakcyjnej musi nadać jej wartość. Dla funkcji oznacza to, że każda podklasa musi zaimplementować treść funkcji.

W Dwellingklasie zdefiniowałeś abstractzmienną buildingMaterial. SquareCabinjest podklasą Dwelling, więc musi zawierać wartość buildingMaterial. Użyj overridesłowa kluczowego, aby wskazać, że ta właściwość została zdefiniowana w klasie nadrzędnej i zostanie zastąpiona w tej klasie.

1. Wewnątrz SquareCabinklasy overridewłaściwość buildingMaterialoraz przypisz jej wartość "Wood".
2. Zrób to samo dla capacity, mówiąc , że 6 mieszkańców może mieszkać w jednym SquareCabin.

```
classSquareCabin(residents:Int):Dwelling(residents){
  overridevalbuildingMaterial="Wood"
  overridevalcapacity=6
}
```

Twój gotowy kod powinien wyglądać tak.

```
abstractclassDwelling(privatevarresidents:Int){
  abstractvalbuildingMaterial:String
  abstractvalcapacity:Int

  funhasRoom():Boolean{
    returnresidents<capacity
  }
}

classSquareCabin(residents:Int):Dwelling(residents){
  overridevalbuildingMaterial="Wood"
  overridevalcapacity=6
}
```

Aby przetestować swój kod, utwórz instancję SquareCabinw swoim programie.

Użyj SquareCabin

1. Wstaw pustą main()funkcję przed definicjami klas Dwellingi .SquareCabin

```
funmain(){

}

abstractclassDwelling(privatevarresidents:Int){
  ...
}

classSquareCabin(residents:Int):Dwelling(residents){
  ...
}
```

1. W ramach main()funkcji utwórz instancję o SquareCabinnazwie squareCabinz 6 mieszkańcami. Dodaj instrukcje drukowania dotyczące materiału budowlanego, pojemności i hasRoom()funkcji.

```
funmain(){
  valsquareCabin=SquareCabin(6)
```

```
    println("\nSquareCabin\n============")
    println("Capacity: ${squareCabin.capacity}")
    println("Material: ${squareCabin.buildingMaterial}")
    println("Has room? ${squareCabin.hasRoom()}")
}
```

Zauważ, że `hasRoom()`funkcja nie została zdefiniowana w `SquareCabin`klasie, ale została zdefiniowana w `Dwelling`klasie. Ponieważ `SquareCabin`jest podklasą `Dwelling`klasy, `hasRoom()`funkcja została odziedziczona za darmo. Funkcja `hasRoom()`może być teraz wywoływana we wszystkich instancjach `SquareCabin`, jak widać we fragmencie kodu jako`squareCabin.hasRoom()`.

1. Uruchom swój kod i powinien wyświetlić następujące informacje.

```
SquareCabin
============
Capacity: 6
Material: Wood
Has room? false
```

Utworzyłeś `squareCabin`z `6`mieszkańcami, co jest równe `capacity`, więc `hasRoom()`zwraca `false`. Możesz poeksperymentować z inicjalizacją `SquareCabin`z mniejszą liczbą `residents`, a po ponownym uruchomieniu programu `hasRoom()`powinien zwrócić `true`.

Użyj z, aby uprościć swój kod

W `println()`instrukcjach, za każdym razem, gdy odwołujesz się do właściwości lub funkcji `squareCabin`, zauważ, jak musisz to powtarzać `squareCabin.`. To staje się powtarzalne i może być źródłem błędów podczas kopiowania i wklejania instrukcji print.

Kiedy pracujesz z określoną instancją klasy i potrzebujesz dostępu do wielu właściwości i funkcji tej instancji, możesz powiedzieć „wykonaj wszystkie poniższe operacje z tym obiektem instancji” za pomocą `with`instrukcji. Zacznij od słowa kluczowego `with`, po którym następuje nazwa instancji w nawiasach, a następnie nawiasy klamrowe zawierające operacje, które chcesz wykonać.

```
with(instanceName){
    // alloperations to do with instanceName
}
```

1. W `main()`funkcji zmień instrukcje print na use `with`.
2. Usuń `squareCabin.`w instrukcjach drukowania.

```
with(squareCabin){
    println("\nSquareCabin\n============")
    println("Capacity: ${capacity}")
    println("Material: ${buildingMaterial}")
    println("Has room? ${hasRoom()}")
}
```

1. Uruchom kod ponownie, aby upewnić się, że działa bez błędów i wyświetla te same dane wyjściowe.

```
SquareCabin
============
Capacity: 6
Material: Wood
Has room? false
```

To jest Twój wypełniony kod:

```
funmain(){
  valsquareCabin=SquareCabin(6)

  with(squareCabin){
    println("\nSquareCabin\n============")
    println("Capacity: ${capacity}")
    println("Material: ${buildingMaterial}")
    println("Has room? ${hasRoom()}")
  }
}


abstractclassDwelling(privatevarresidents:Int){
  abstractvalbuildingMaterial:String
  abstractvalcapacity:Int

  funhasRoom():Boolean{
    returnresidents<capacity
  }
}

classSquareCabin(residents:Int):Dwelling(residents){
  overridevalbuildingMaterial="Wood"
  overridevalcapacity=6
}
```

Utwórz podklasę RoundHut

1. W taki sam sposób jak `SquareCabin`, dodaj kolejną podklasę `RoundHut`, do `Dwelling`.
2. Zastąp `buildingMaterial`i nadaj mu wartość `"Straw"`.
3. Zastąp `capacity`i ustaw na 4.

```
classRoundHut(residents:Int):Dwelling(residents){
  overridevalbuildingMaterial="Straw"
  overridevalcapacity=4
}
```

1. W `main()`programie utwórz instancję `RoundHut`z 3 mieszkańcami.

```
valroundHut=RoundHut(3)
```

1. Dodaj poniższy kod, aby wydrukować informacje o `roundHut`.

```
with(roundHut){
  println("\nRound Hut\n=========")
  println("Material: ${buildingMaterial}")
  println("Capacity: ${capacity}")
  println("Has room? ${hasRoom()}")
}
```

1. Uruchom swój kod, a wynik dla całego programu powinien wyglądać tak:

```
SquareCabin
============
Capacity: 6
Material: Wood
Has room? false

Round Hut
=========
Material: Straw
Capacity: 4
Has room? true
```

Masz teraz hierarchię klas, która wygląda tak, z `Dwelling`klasą główną `SquareCabin`i `RoundHut`podklasami `Dwelling`.

Utwórz podklasę RoundTower

Ostatnią klasą w tej hierarchii klas jest okrągła wieża. Możesz myśleć o okrągłej wieży jako okrągłej chacie wykonanej z kamienia, z wieloma piętrami. Możesz więc utworzyć `RoundTower`podklasę `RoundHut`.

1. Utwórz `RoundTower`klasę, która jest podklasą `RoundHut`. Dodaj `residents`parametr do konstruktora `RoundTower`, a następnie przekaż ten parametr do konstruktora `RoundHut`nadklasy.
2. Zastąp `buildingMaterial`to być `"Stone"`.
3. Ustaw `capacity`na `4`.

```
classRoundTower(residents:Int):RoundHut(residents){
  overridevalbuildingMaterial="Stone"
  overridevalcapacity=4
```

```
}
```

1. Uruchom ten kod, a otrzymasz błąd.

```
Thistypeisfinal, soitcannot be inherited from
```

Ten błąd oznacza, że `RoundHut`klasa nie może być podklasowana (ani dziedziczona). Domyślnie w Kotlinie klasy są ostateczne i nie mogą być dzielone na podklasy. Możesz dziedziczyć tylko z `abstract`klas lub klas, które są oznaczone `open`słowem kluczowym. Dlatego musisz oznaczyć `RoundHut`klasę `open`słowem kluczowym, aby umożliwić jej dziedziczenie.

Uwaga: nie musisz używać `open`słowa kluczowego podczas definiowania klas abstrakcyjnych . Przykład: `Dwelling`klasa nie musi być oznaczona `open`słowem kluczowym. Zakłada się, że będziesz chciał utworzyć podklasę klasy abstrakcyjnej, aby zaimplementować abstrakcyjne właściwości i funkcje.

1. Dodaj `open`słowo kluczowe na początku `RoundHut`deklaracji.

```
open classRoundHut(residents:Int):Dwelling(residents){
  overridevalbuildingMaterial="Straw"
  overridevalcapacity=4
}
```

1. W `main()`programie utwórz wystąpienie `roundTower`i wydrukuj informacje o nim.

```
 valroundTower=RoundTower(4)

with(roundTower){
   println("\nRound Tower\n==========")
   println("Material: ${buildingMaterial}")
   println("Capacity: ${capacity}")
   println("Has room? ${hasRoom()}")
}
```

Oto kompletny kod.

```
funmain(){
   valsquareCabin=SquareCabin(6)
   valroundHut=RoundHut(3)
   valroundTower=RoundTower(4)

   with(squareCabin){
      println("\nSquareCabin\n============")
      println("Capacity: ${capacity}")
      println("Material: ${buildingMaterial}")
      println("Has room? ${hasRoom()}")
   }

   with(roundHut){
```

```
      println("\nRound Hut\n=========")
      println("Material: ${buildingMaterial}")
      println("Capacity: ${capacity}")
      println("Has room? ${hasRoom()}")
   }

   with(roundTower){
      println("\nRound Tower\n==========")
      println("Material: ${buildingMaterial}")
      println("Capacity: ${capacity}")
      println("Has room? ${hasRoom()}")
   }
}


abstractclassDwelling(privatevarresidents:Int){
   abstractvalbuildingMaterial:String
   abstractvalcapacity:Int

   funhasRoom():Boolean{
     returnresidents<capacity
  }
}

classSquareCabin(residents:Int):Dwelling(residents){
   overridevalbuildingMaterial="Wood"
   overridevalcapacity=6
}

open classRoundHut(residents:Int):Dwelling(residents){
  overridevalbuildingMaterial="Straw"
  overridevalcapacity=4
}

classRoundTower(residents:Int):RoundHut(residents){
   overridevalbuildingMaterial="Stone"
   overridevalcapacity=4
}
```

1. Uruchom swój kod. Powinien teraz działać bez błędów i generować następujące dane wyjściowe.

```
SquareCabin
============
Capacity: 6
Material: Wood
Has room? false
```

Round Hut
=========
Material: Straw
Capacity: 4
Has room? true

Round Tower
==========
Material: Stone
Capacity: 4
Has room? false
Dodaj wiele pięter do RoundTower

RoundHut, co sugeruje, jest budynkiem jednopiętrowym. Wieże mają zwykle wiele kondygnacji (pięter).

Myśląc o pojemności, im więcej pięter ma wieża, tym większą powinna mieć pojemność.

Możesz zmodyfikować RoundTower, aby mieć wiele pięter, i dostosować jego pojemność na podstawie liczby pięter.

1. Zaktualizuj RoundTowerkonstruktor, aby uwzględnić dodatkowy parametr liczby całkowitej valfloorsdla liczby pięter. Umieść to po residents. Zauważ, że nie musisz przekazywać tego do RoundHutkonstruktora nadrzędnego, ponieważ floorsjest zdefiniowany tutaj w RoundTowerі RoundHutnie ma floors.

```
classRoundTower(
  residents:Int,
  valfloors:Int):RoundHut(residents){

  …
}
```

Uwaga: Gdy masz wiele parametrów, a definicja klasy lub funkcji staje się zbyt długa, aby wygodnie zmieścić się w jednym wierszu, możesz podzielić ją na wiele wierszy, jak pokazano w powyższym kodzie.

1. Uruchom swój kod. Podczas tworzenia roundTowerw main()metodzie wystąpił błąd, ponieważ nie podajesz liczby dla floorsargumentu. Możesz dodać brakujący argument.

Alternatywnie, w definicji klasy RoundTower, możesz dodać domyślną wartość dla floors, jak pokazano poniżej. Następnie, gdy floorsdo konstruktora nie zostanie przekazana żadna wartość for, do utworzenia instancji obiektu można użyć wartości domyślnej.

1. W swoim kodzie dodaj = 2po deklaracji , floorsaby przypisać mu domyślną wartość 2.

```
classRoundTower(
  residents:Int,
  valfloors:Int=2):RoundHut(residents){

  …
}
```

1. Uruchom swój kod. Powinien się skompilować, ponieważ `RoundTower(4)`teraz tworzy `RoundTower`instancję obiektu z domyślną wartością 2 pięter.
2. W `RoundTower`klasie zaktualizuj wartość, `capacity`aby pomnożyć ją przez liczbę pięter.

```
overridevalcapacity=4*floors
```

1. Uruchom kod i zauważ, że `RoundTower`pojemność wynosi teraz 8 na 2 piętra.

Oto Twój gotowy kod.

```
funmain(){

  valsquareCabin=SquareCabin(6)
  valroundHut=RoundHut(3)
  valroundTower=RoundTower(4)

  with(squareCabin){
    println("\nSquareCabin\n============")
    println("Capacity: ${capacity}")
    println("Material: ${buildingMaterial}")
    println("Has room? ${hasRoom()}")
  }

  with(roundHut){
    println("\nRound Hut\n=========")
    println("Material: ${buildingMaterial}")
    println("Capacity: ${capacity}")
    println("Has room? ${hasRoom()}")
  }

  with(roundTower){
    println("\nRound Tower\n==========")
    println("Material: ${buildingMaterial}")
    println("Capacity: ${capacity}")
    println("Has room? ${hasRoom()}")
  }
}


abstractclassDwelling(privatevarresidents:Int){
  abstractvalbuildingMaterial:String
  abstractvalcapacity:Int

  funhasRoom():Boolean{
    returnresidents<capacity
  }
}
```

```
classSquareCabin(residents:Int):Dwelling(residents){
  overridevalbuildingMaterial="Wood"
  overridevalcapacity=6
}


open classRoundHut(residents:Int):Dwelling(residents){
  overridevalbuildingMaterial="Straw"
  overridevalcapacity=4
}


classRoundTower(
  residents:Int,
  valfloors:Int=2):RoundHut(residents){

  overridevalbuildingMaterial="Stone"
  overridevalcapacity=4*floors
}
```

5. Modyfikuj klasy w hierarchii

Oblicz powierzchnię podłogi

W tym ćwiczeniu dowiesz się, jak zadeklarować funkcję abstrakcyjną w klasie abstrakcyjnej, a następnie zaimplementować jej funkcjonalność w podklasach.

Wszystkie mieszkania mają powierzchnię użytkową, jednak w zależności od kształtu mieszkania obliczana jest inaczej.

Zdefiniuj floorArea() w klasie Dwelling

1. Najpierw dodaj `abstract floorArea()`funkcję do `Dwelling`klasy. Zwróć `Double`. Double to typ danych, jak `String`i `Int`; jest używany do liczb zmiennoprzecinkowych, czyli liczb, które mają kropkę dziesiętną, po której następuje część ułamkowa, np. 5.8793.)

```
abstractfunfloorArea():Double
```

Wszystkie metody abstrakcyjne zdefiniowane w klasie abstrakcyjnej muszą być zaimplementowane w dowolnej z jej podklas. Zanim będziesz mógł uruchomić swój kod, musisz zaimplementować `floorArea()`w podklasach.

Implementuj floorArea() dla SquareCabin

Podobnie jak w przypadku `buildingMaterial`i `capacity`, ponieważ implementujesz `abstract`funkcję zdefiniowaną w klasie nadrzędnej, musisz użyć `override`słowa kluczowego.

1. W `SquareCabin`klasie zacznij od słowa kluczowego `override`, po którym następuje rzeczywista implementacja `floorArea()`funkcji, jak pokazano poniżej.

```
overridefunfloorArea():Double{

}
```

1. Zwróć obliczoną powierzchnię podłogi. Pole powierzchni prostokąta lub kwadratu to długość jego boku pomnożona przez długość jego drugiego boku. Treść funkcji będzie `return length * length`.

```
overridefunfloorArea():Double{
  returnlength*length
}
```

Długość nie jest zmienną w klasie i jest inna dla każdej instancji, więc można ją dodać jako parametr konstruktora dla `SquareCabin`klasy.

1. Zmień definicję klasy, `SquareCabin`aby dodać `length`parametr typu `Double`. Zadeklaruj nieruchomość jako , `val`ponieważ długość budynku się nie zmienia.

```
classSquareCabin(residents:Int,vallength:Double):Dwelling(residents){
```

`Dwelling`i dlatego wszystkie jego podklasy mają `residents`jako argument konstruktora. Ponieważ jest to pierwszy argument w `Dwelling`konstruktorze, najlepszą praktyką jest ustawienie go również jako pierwszego argumentu we wszystkich konstruktorach podklas i umieszczenie argumentów w tej samej kolejności we wszystkich definicjach klas. Dlatego wstaw nowy `length`parametr po `residents`parametrze.

1. W `main()`aktualizacji tworzenie `squareCabin`instancji. Przekaż `50.0`do `SquareCabin`konstruktora jako `length`.

```
valsquareCabin=SquareCabin(6,50.0)
```

1. Wewnątrz `with`wyrażenia for `squareCabin`dodaj wyrażenie print dotyczące powierzchni podłogi.

```
println("Floorarea: ${floorArea()}")
```

Twój kod się nie uruchomi, ponieważ musisz również zaimplementować `floorArea()`w `RoundHut`.

Implementuj floorArea() dla RoundHut

W ten sam sposób zaimplementuj powierzchnię podłogi dla `RoundHut`. `RoundHut`jest również bezpośrednią podklasą `Dwelling`, więc musisz użyć `override`słowa kluczowego.

Powierzchnia kondygnacji mieszkania kołowego to PI * promień^2 lub PI * promień * promień.

`PI`jest wartością matematyczną. Jest zdefiniowany w bibliotece matematycznej. Biblioteka to predefiniowana kolekcja funkcji i wartości zdefiniowanych poza programem, z której program może korzystać. Aby użyć funkcji lub wartości bibliotecznej, musisz poinformować kompilator, że zamierzasz jej użyć. Robisz to, importując funkcję lub wartość do swojego programu. Aby użyć `PI`w swoim programie, musisz zaimportować `kotlin.math.PI`.

1. Importuj `PI`z biblioteki matematycznej Kotlin. Umieść to na początku pliku, przed `main()`.

```
importkotlin.math.PI
```

1. Zaimplementuj floorArea()funkcję dla RoundHut.

```
overridefunfloorArea():Double{
   return PI * radius * radius
}
```

Ostrzeżenie: jeśli nie importujesz

```
kotlin.math.PI
```

, pojawi się błąd, więc zaimportuj tę bibliotekę przed jej użyciem. Alternatywnie możesz napisać w pełni kwalifikowaną wersję

```
PI
```

, jak w

```
kotlin.math.PI* radius * radius
```

, a następnie instrukcja import nie jest potrzebna.

1. Zaktualizuj RoundHutkonstruktora, aby przekazać radius.

```
open classRoundHut(
  valresidents:Int,
  val radius:Double):Dwelling(residents){
```

1. W main()programie zaktualizuj inicjalizację roundHut, przekazując a do radiuskonstruktora .10.0RoundHut

```
valroundHut=RoundHut(3,10.0)
```

1. Dodaj instrukcję print wewnątrz withinstrukcji for roundHut.

```
println("Floorarea: ${floorArea()}")
```

Implementuj floorArea() dla RoundTower

Twój kod jeszcze się nie uruchamia i kończy się niepowodzeniem z następującym błędem:

```
Error: No valuepassed for parameter 'radius'
```

W RoundTower, aby Twój program się skompilował, nie musisz implementować floorArea(), ponieważ jest dziedziczony z RoundHut, ale musisz zaktualizować RoundTowerdefinicję klasy, aby miała również ten sam radiusargument, co jej rodzicRoundHut.

1. Zmień konstruktora RoundTower na radius. Umieść radiuspo residentsi przed floors. Zaleca się, aby zmienne z wartościami domyślnymi były wymienione na końcu. Pamiętaj o przekazaniu radiusdo konstruktora klasy nadrzędnej.

```
classRoundTower(
  residents:Int,
  radius:Double,
  valfloors:Int=2):RoundHut(residents, radius){
```

1. Zaktualizuj inicjalizację `roundTower`w `main()`.

```
valroundTower=RoundTower(4,15.5)
```

1. I dodaj instrukcję print, która wywołuje `floorArea()`.

```
println("Floorarea: ${floorArea()}")
```

1. Możesz teraz uruchomić swój kod!
2. Zauważ, że obliczenie dla `RoundTower`nie jest poprawne, ponieważ jest dziedziczone z `RoundHut`i nie uwzględnia liczby `floors`.
3. W `RoundTower`, `overridefloorArea()`można więc nadać mu inną implementację, która zwielokrotnia powierzchnię przez liczbę pięter. Zwróć uwagę, jak możesz zdefiniować funkcję w klasie abstrakcyjnej (`Dwelling`), zaimplementować ją w podklasie (`RoundHut`), a następnie ponownie ją zastąpić w podklasie podklasy (`RoundTower`). To najlepsze z obu światów — dziedziczysz pożądaną funkcjonalność i możesz zastąpić tę, której nie chcesz.

```
overridefunfloorArea():Double{
  return PI * radius * radius *floors
}
```

Ten kod działa, ale istnieje sposób na uniknięcie powtarzania kodu, który jest już w `RoundHut`klasie nadrzędnej. Możesz wywołać `floorArea()`funkcję z klasy nadrzędnej `RoundHut`, która zwraca `PI * radius * radius`.Then pomnóż ten wynik przez liczbę `floors`.

1. W programie `RoundTower`, zaktualizuj, `floorArea()`aby użyć implementacji superklasy `floorArea()`. Użyj `super`słowa kluczowego, aby wywołać funkcję zdefiniowaną w rodzicu.

```
overridefunfloorArea():Double{
  returnsuper.floorArea()*floors
}
```

1. Uruchom kod ponownie i `RoundTower`wypisuje odpowiednią powierzchnię dla wielu pięter.

Oto Twój gotowy kod:

```
importkotlin.math.PI

funmain(){

  valsquareCabin=SquareCabin(6,50.0)
  valroundHut=RoundHut(3,10.0)
  valroundTower=RoundTower(4,15.5)
```

```
    with(squareCabin){
        println("\nSquareCabin\n============")
        println("Capacity: ${capacity}")
        println("Material: ${buildingMaterial}")
        println("Has room? ${hasRoom()}")
        println("Floorarea: ${floorArea()}")
    }

    with(roundHut){
        println("\nRound Hut\n=========")
        println("Material: ${buildingMaterial}")
        println("Capacity: ${capacity}")
        println("Has room? ${hasRoom()}")
        println("Floorarea: ${floorArea()}")
    }

    with(roundTower){
        println("\nRound Tower\n==========")
        println("Material: ${buildingMaterial}")
        println("Capacity: ${capacity}")
        println("Has room? ${hasRoom()}")
        println("Floorarea: ${floorArea()}")
    }
}


abstractclassDwelling(privatevarresidents:Int){

    abstractvalbuildingMaterial:String
    abstractvalcapacity:Int

    funhasRoom():Boolean{
        returnresidents<capacity
}

    abstractfunfloorArea():Double
}


classSquareCabin(residents:Int,
    vallength:Double):Dwelling(residents){

    overridevalbuildingMaterial="Wood"
    overridevalcapacity=6
```

```
  overridefunfloorArea():Double{
    returnlength*length
  }
}


open classRoundHut(valresidents:Int,
  val radius:Double):Dwelling(residents){

  overridevalbuildingMaterial="Straw"
  overridevalcapacity=4

  overridefunfloorArea():Double{
    return PI * radius * radius
  }
}


classRoundTower(residents:Int, radius:Double,
  valfloors:Int=2):RoundHut(residents, radius){

  overridevalbuildingMaterial="Stone"
  overridevalcapacity=4*floors

  overridefunfloorArea():Double{
    returnsuper.floorArea()*floors
  }
}
```

Dane wyjściowe powinny być:

```
SquareCabin
============
Capacity: 6
Material: Wood
Has room? false
Floorarea: 2500.0

Round Hut
=========
Material: Straw
Capacity: 4
Has room? true
Floorarea: 314.1592653589793

Round Tower
```

==========
Material: Stone
Capacity: 8
Has room? true
Floorarea: 1509.5352700498956

Uwaga: w przypadku wartości obszaru przyjemniejszym dla użytkownika byłoby pokazanie tylko kilku miejsc po przecinku. To jest poza zakresem tego ćwiczenia z programowania, ale możesz wydrukować powierzchnię podłogi za pomocą tego: println("Powierzchnia podłogi: %.2f".format(floorArea()))

Pozwól nowemu mieszkańcowi dostać pokój

Dodaj możliwość, aby nowy mieszkaniec otrzymał pokój z `getRoom()`funkcją zwiększającą liczbę mieszkańców o jednego. Ponieważ ta logika jest taka sama dla wszystkich mieszkań, możesz zaimplementować funkcję w `Dwelling`, a to sprawi, że będzie ona dostępna dla wszystkich podklas i ich dzieci. Schludny!

Uwagi:

- Użyj `if`stwierdzenia, które dodaje mieszkańca tylko wtedy, gdy jest jeszcze wolne miejsce.
- Wydrukuj wiadomość z wynikiem.
- Możesz użyć `residents++`jako skrótu dla `residents = residents + 1`, aby dodać 1 do `residents`zmiennej.

1. Zaimplementuj `getRoom()`funkcję w `Dwelling`klasie.

```
fungetRoom(){
  if(capacity>residents){
    residents++
    println("Yougot a room!")
  }else{
    println("Sorry, atcapacity and no roomsleft.")
  }
}
```

1. Dodaj kilka instrukcji print do `with`bloku instrukcji, `roundHut`aby zaobserwować, co dzieje się `getRoom()`i `hasRoom()`używane razem.

```
println("Has room? ${hasRoom()}")
getRoom()
println("Has room? ${hasRoom()}")
getRoom()
```

Dane wyjściowe dla tych instrukcji drukowania:

Has room? true
Yougot a room!
Has room? false
Sorry, atcapacity and no roomsleft.

Zobacz kod rozwiązania, aby uzyskać szczegółowe informacje.

Uwaga: to jest przykład tego, dlaczego dziedziczenie jest tak potężne. Możesz wywołać `getRoom()`funkcję na wszystkich podklasach `Dwelling`bez dodatkowego kodu w tych klasach.

Dopasuj dywan do okrągłego mieszkania

Powiedzmy, że musisz wiedzieć, jaki rozmiar dywanu kupić dla swojego `RoundHut`lub `RoundTower`. Dla `SquareCabin`, maksymalna długość dywanu jest taka sama jak powierzchnia podłogi z długości, więc nie są potrzebne żadne dodatkowe obliczenia. Włącz funkcję `RoundHut`, aby była dostępna dla wszystkich mieszkań okrągłych.

1. Najpierw zaimportuj `sqrt()`funkcję z `kotlin.math`biblioteki.

```
importkotlin.math.sqrt
```

1. Zaimplementuj `calculateMaxCarpetSize()`funkcję w `RoundHut`klasie. Wzór na dopasowanie prostokąta do koła to pierwiastek kwadratowy ze kwadratowej średnicy podzielonej przez 2:`sqrt (diameter * diameter / 2)`

```
funcalculateMaxCarpetSize():Double{
  valdiameter=2* radius
  returnsqrt(diameter*diameter/2)
}
```

1. `calculateMaxCarpetSize()`Teraz można wywoływać metodę i instancje `RoundHut`. `RoundTower`Dodaj instrukcje print do `roundHut`i `roundTower`w `main()`funkcji.

```
println("Carpetsize: ${calculateMaxCarpetSize()}")
```

Zobacz kod rozwiązania, aby uzyskać szczegółowe informacje.

Uwaga: zwróć uwagę, że dodałeś `calculateMaxCarpetSize()`do `RoundHut`i `RoundTower`dziedziczy go. Nie możesz jednak wywołać tej funkcji na `squareCabin`instancji, ponieważ jej nie definiuje lub ma nadklasę, która ją definiuje.

Gratulacje! Stworzyłeś kompletną hierarchię klas z właściwościami i funkcjami, ucząc się wszystkiego, czego potrzebujesz, aby tworzyć bardziej przydatne klasy!

6. Kod rozwiązania

To jest kompletny kod rozwiązania dla tego ćwiczenia z programowania, w tym komentarze.

```
/**
* Program thatimplementsclasses for differentkinds of dwellings.
* Showshow to:
* Createclass hierarchy, variables and functions with inheritance,
```

```
* abstractclass, overriding, and private vs. public variables.
*/

importkotlin.math.PI
importkotlin.math.sqrt

funmain(){
  valsquareCabin=SquareCabin(6,50.0)
  valroundHut=RoundHut(3,10.0)
  valroundTower=RoundTower(4,15.5)

  with(squareCabin){
    println("\nSquareCabin\n============")
    println("Capacity: ${capacity}")
    println("Material: ${buildingMaterial}")
    println("Floorarea: ${floorArea()}")
  }

  with(roundHut){
    println("\nRound Hut\n=========")
    println("Material: ${buildingMaterial}")
    println("Capacity: ${capacity}")
    println("Floorarea: ${floorArea()}")
    println("Has room? ${hasRoom()}")
    getRoom()
    println("Has room? ${hasRoom()}")
    getRoom()
    println("Carpetsize: ${calculateMaxCarpetSize()}")
  }

  with(roundTower){
    println("\nRound Tower\n==========")
    println("Material: ${buildingMaterial}")
    println("Capacity: ${capacity}")
    println("Floorarea: ${floorArea()}")
    println("Carpetsize: ${calculateMaxCarpetSize()}")
  }
}


/**
* Definespropertiescommon to alldwellings.
* Alldwellingshavefloorspace,
* but itscalculationisspecific to the subclass.
* Checking and getting a roomareimplementedhere
* becausetheyare the same for allDwellingsubclasses.
```

```
*
* @param residentsCurrentnumber of residents
*/
abstractclassDwelling(privatevarresidents:Int){
  abstractvalbuildingMaterial:String
  abstractvalcapacity:Int

  /**
   * Calculates the floorarea of the dwelling.
   * Implemented by subclasseswhereshapeisdetermined.
   *
   * @return floorarea
   */
  abstractfunfloorArea():Double

  /**
   * Checkswhetherthereisroom for anotherresident.
   *
   * @return trueifroomavailable, falseotherwise
   */
  funhasRoom():Boolean{
    returnresidents<capacity
  }

  /**
   * Compares the capacity to the number of residents and
   * ifcapacityislargerthannumber of residents,
   * addresident by increasing the number of residents.
   * Print the result.
   */
  fungetRoom(){
    if(capacity>residents){
      residents++
      println("Yougot a room!")
    }else{
      println("Sorry, atcapacity and no roomsleft.")
    }
  }

  }

/**
* A squarecabindwelling.
*
*  @param residentsCurrentnumber of residents
*  @param lengthLength
```

```
*/
classSquareCabin(residents:Int,vallength:Double):Dwelling(residents){
  overridevalbuildingMaterial="Wood"
  overridevalcapacity=6

  /**
   * Calculatesfloorarea for a squaredwelling.
   *
   * @return floorarea
   */
  overridefunfloorArea():Double{
     returnlength*length
  }

}

/**
* Dwelling with a circularfloorspace
*
* @param residentsCurrentnumber of residents
* @param radius Radius
*/
open classRoundHut(
     valresidents:Int,val radius:Double):Dwelling(residents){

  overridevalbuildingMaterial="Straw"
  overridevalcapacity=4

  /**
   * Calculatesfloorarea for a rounddwelling.
   *
   * @return floorarea
   */
  overridefunfloorArea():Double{
     return PI * radius * radius
  }

  /**
   *  Calculates the max length for a squarecarpet
   *  thatfits the circularfloor.
   *
   * @return length of carpet
   */
  funcalculateMaxCarpetSize():Double{
     valdiameter=2* radius
     returnsqrt(diameter*diameter/2)
```

```
  }

}

/**
* Roundtower with multiplestories.
*
* @param residentsCurrentnumber of residents
* @param radius Radius
* @param floorsNumber of stories
*/
classRoundTower(
    residents:Int,
    radius:Double,
    valfloors:Int=2):RoundHut(residents, radius){

  overridevalbuildingMaterial="Stone"

  // Capacitydepends on the number of floors.
  overridevalcapacity=floors*4

  /**
   * Calculates the totalfloorarea for a towerdwelling
   * with multiplestories.
   *
   * @return floorarea
   */
  overridefunfloorArea():Double{
    returnsuper.floorArea()*floors
  }
}
```

7. Podsumowanie

Podczas tego ćwiczenia z programowania nauczyłeś się:

- Utwórz hierarchię klas, czyli drzewo klas, w którym dzieci dziedziczą funkcjonalność klas nadrzędnych. Właściwości i funkcje są dziedziczone przez podklasy.
- Utwórz `abstract`klasę, w której część funkcjonalności pozostaje do zaimplementowania przez jej podklasy. `abstract`Dlatego nie można utworzyć instancji klasy .
- Utwórz podklasy `abstract`klasy.
- Użyj `override`słowa kluczowego, aby zastąpić właściwości i funkcje w podklasach.
- Użyj `super`słowa kluczowego, aby odwołać się do funkcji i właściwości w klasie nadrzędnej.
- Utwórz klasę `open`, aby można ją było podzielić na podklasy.
- Utwórz właściwość `private`, aby można było jej używać tylko wewnątrz klasy.

- Użyj `with`konstrukcji, aby wykonać wiele wywołań tego samego wystąpienia obiektu.
- Importuj funkcjonalność z `kotlin.math`biblioteki

8. Dowiedz się więcej

- Dziedzictwo
- Klasy i dziedziczenie
- Konstruktorzy
- Dziedziczenie (abstrakcyjne, otwarte, nadpisane, prywatne)
- `with`(formalna definicja)

Twórz układy XML dla Androida

1. Zanim zaczniesz

Podczas tego ćwiczenia z programowania zbudujesz układ podstawowej aplikacji kalkulatora napiwków. Pod koniec ćwiczenia z programowania będziesz mieć działający interfejs użytkownika aplikacji, ale aplikacja nie obliczy jeszcze napiwku. Aby aplikacja działała i wyglądała bardziej profesjonalnie, zajmiemy się poniższymi ćwiczeniami z programowania.

Warunki wstępne

- Możliwość tworzenia i uruchamiania aplikacji na Androida z szablonu w Android Studio

Czego się nauczysz

- Jak czytać i pisać układy XML w Androidzie
- Jak zbudować układ prostego formularza do wprowadzania tekstu i wyborów użytkownika

Co zbudujesz

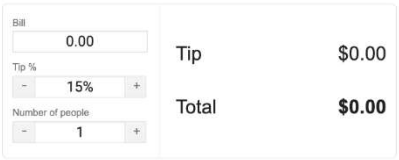
- Interfejs użytkownika aplikacji kalkulatora napiwków na Androida

Czego potrzebujesz

- Komputer z zainstalowaną najnowszą stabilną wersją Android Studio
- Połączenie internetowe w celu uzyskania dostępu do dokumentacji dla programistów Androida

2. Rozpocznij projekt

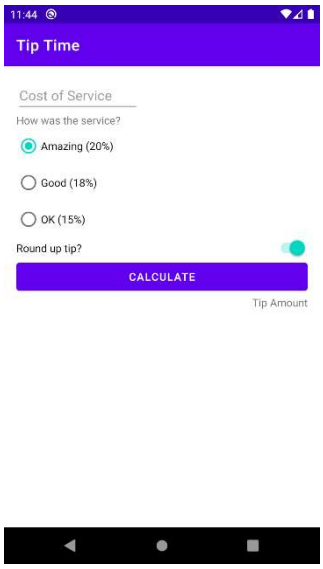
Sprawdź kalkulator napiwków w Google: https://www.google.com/search?q=tip+calculator



W tej ścieżce zbudujesz prostą wersję kalkulatora napiwków jako aplikację na Androida.

Deweloperzy często pracują w ten sposób — przygotowując prostą wersję aplikacji gotową i częściowo działającą (nawet jeśli nie wygląda zbyt dobrze), a następnie dopracowując ją w pełni funkcjonalną i dopracowaną wizualnie później.

Pod koniec tego ćwiczenia z programowania aplikacja kalkulatora napiwków będzie wyglądać tak:

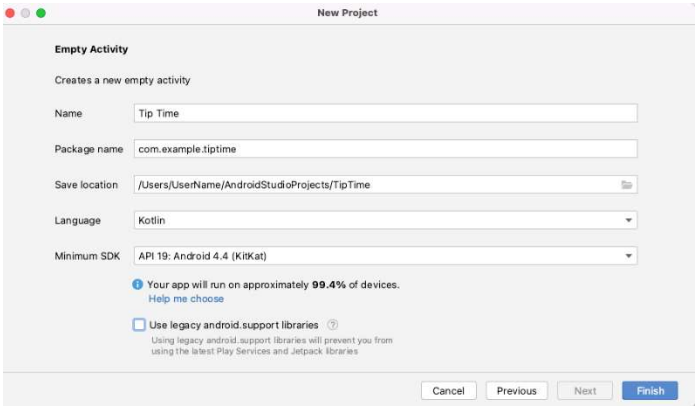


Będziesz używać tych elementów interfejsu użytkownika dostarczanych przez system Android:

- EditText- do wprowadzania i edycji tekstu
- TextView- aby wyświetlić tekst, taki jak pytanie serwisowe i kwota napiwku
- RadioButton- przycisk radiowy do wyboru dla każdej opcji końcówki
- RadioGroup- aby pogrupować opcje przycisków radiowych
- Switch- przełącznik włączania/wyłączania do wyboru, czy zaokrąglić końcówkę, czy nie

Utwórz projekt pustej aktywności

1. Aby rozpocząć, utwórz nowy projekt Kotlin w Android Studio, korzystając z szablonu **Empty Activity .**
2. Wywołaj aplikację „Tip Time” z minimalnym poziomem interfejsu API 19 (KitKat). Nazwa pakietu to **com.example.tiptime** .



1. Kliknij **Zakończ** , aby utworzyć aplikację.

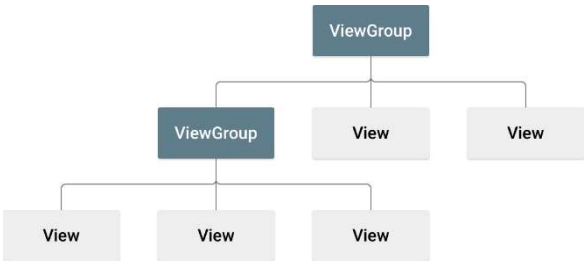
3. Przeczytaj i zrozum XML

Zamiast korzystać z **Edytora układu** , który już znasz, zbudujesz układ swojej aplikacji, modyfikując kod XML opisujący interfejs użytkownika. Nauczenie się, jak rozumieć i modyfikować układy interfejsu użytkownika za pomocą XML, będzie ważne dla Ciebie jako programisty Androida.

Będziesz przeglądać i edytować plik XML, który definiuje układ interfejsu użytkownika dla tej aplikacji. XML to skrót od *eXtensibleMarkup Language* , który jest sposobem opisywania danych za pomocą dokumentu tekstowego. Ponieważ XML jest rozszerzalny i bardzo elastyczny, jest używany do wielu różnych rzeczy, w tym do definiowania układu interfejsu użytkownika aplikacji na Androida. Możesz przypomnieć sobie z wcześniejszych ćwiczeń z programowania, że inne zasoby, takie jak ciągi dla Twojej aplikacji, są również zdefiniowane w pliku XML o nazwie `strings.xml`.

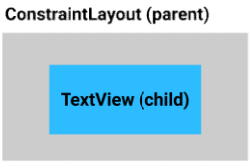
Interfejs użytkownika aplikacji dla systemu Android jest zbudowany jako hierarchia zawierająca składniki (widżety) i układy ekranowe tych składników. Zauważ, że te układy same w sobie są składnikami interfejsu użytkownika.

Opisujesz hierarchię widoków elementów interfejsu użytkownika na ekranie. Na przykład `ConstraintLayout`(nadrzędny) może zawierać `Buttons`, `TextViews`, `ImageViews`lub inne widoki (dzieci). Pamiętaj, `ConstraintLayout`jest podklasą `ViewGroup`. Umożliwia elastyczne ustawienie lub rozmiar widoków podrzędnych.



Hierarchia przechowywania aplikacji na Androida

UWAGA: Widoczna hierarchia interfejsu użytkownika jest oparta na ograniczaniu, tj. jeden komponent zawiera jeden lub więcej komponentów. Nie jest to związane z hierarchią klas i podklas, których nauczyłeś się wcześniej. Terminy rodzic i dziecko są czasami używane, ale kontekst tutaj mówi o widokach nadrzędnych (grupach widoków) zawierających widoki potomne, które z kolei mogą zawierać widoki potomne.

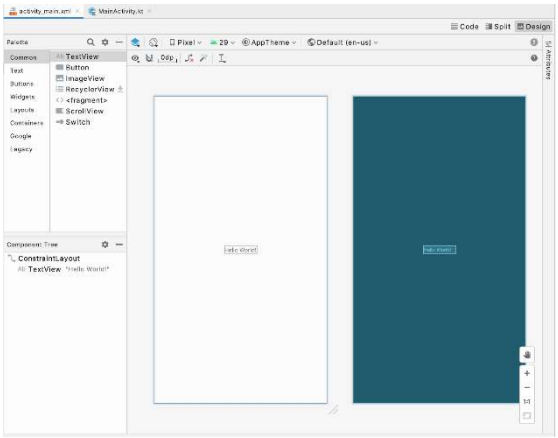


Każdy element interfejsu użytkownika jest reprezentowany przez *element* XML w pliku XML. Każdy element zaczyna się i kończy tagiem, a każdy tag zaczyna się `<`i kończy na `>`. Tak jak możesz ustawić atrybuty w elementach interfejsu użytkownika za pomocą **Edytora układu (projekt)** , elementy XML również mogą mieć *atrybuty* . W uproszczeniu, XML dla powyższych elementów interfejsu użytkownika może wyglądać mniej więcej tak:

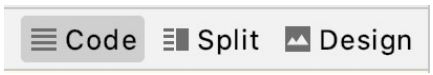
```
<ConstraintLayout>
  <TextView
    text="Hello World!">
  </TextView>
</ConstraintLayout>
```

Spójrzmy na prawdziwy przykład.

1. Otwórz `activity_main.xml`(**res** > **layout** > **activity_main.xml**).
2. Możesz zauważyć, że aplikacja wyświetla komunikat `TextView`„Hello World!” w ramach `ConstraintLayout`, jak widzieliście w poprzednich projektach utworzonych z tego szablonu.



1. Znajdź opcje widoków **Kod** , **Podział** i **Projekt** w prawym górnym rogu Edytora układu.
2. Wybierz widok **Kod** .



`activity_main.xml` Tak wygląda XML w :

```
<?xml version="1.0"encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
  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=".MainActivity">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

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

Dzieje się o *wiele* więcej niż w uproszczonym przykładzie, ale Android Studio robi kilka rzeczy, aby uczynić XML bardziej czytelnym, tak samo jak w przypadku kodu Kotlin.

1. Zwróć uwagę na wcięcie. Android Studio robi to automatycznie, aby pokazać hierarchię elementów. Jest `TextView`wcięty, ponieważ jest zawarty w `ConstraintLayout`. Jest `ConstraintLayout`rodzicem, a `TextView`jest dzieckiem. Atrybuty każdego elementu są wcięte, aby pokazać, że są częścią tego elementu.
2. Zwróć uwagę na kodowanie kolorami — niektóre rzeczy są niebieskie, inne zielone i tak dalej. Podobne części pliku są narysowane w tym samym kolorze, aby ułatwić ich dopasowanie. W szczególności zauważ, że Android Studio rysuje początek i koniec tagów elementów w tym samym kolorze. (Uwaga: kolory używane w ćwiczeniach z programowania mogą nie odpowiadać kolorom widocznym w Android Studio.)

Tagi, elementy i atrybuty XML

Oto uproszczona wersja `TextView`elementu, dzięki czemu możesz przyjrzeć się niektórym ważnym częściom:

```
<TextView
  android:text="Hello World!"
/>
```

Linia with `<TextView`to początek tagu, a linia with `/>`to koniec tagu. Linia z `android:text="Hello World!"`jest atrybutem tagu. Reprezentuje tekst, który będzie wyświetlany przez `TextView`. Te 3 wiersze są powszechnie używanym skrótem zwanym *tagiem pustego elementu* . Oznaczałoby to to samo, gdybyś napisał to z oddzielnym *tagiem początkowym* i *końcowym* , w ten sposób:

```
<TextView
  android:text="Hello World!"
```

```
></TextView>
```

Często zdarza się również, że znacznik pustego elementu zapisuje go w jak najmniejszej liczbie wierszy i łączy koniec znacznika z wierszem przed nim. Możesz więc zobaczyć tag pustego elementu w dwóch wierszach (lub nawet w jednym wierszu, jeśli nie ma atrybutów):

```
<!-- with attributes, two lines -->
<TextView
  android:text="Hello World!"/>
```

Element `ConstraintLayout`jest napisany z oddzielnymi znacznikami początkowymi i końcowymi, ponieważ musi być w stanie pomieścić w sobie inne elementy. Oto uproszczona wersja `ConstraintLayout`elementu z `TextView`elementem w środku:

```
<androidx.constraintlayout.widget.ConstraintLayout>
  <TextView
    android:text="Hello World!"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Jeśli chciałbyś dodać inny `View`jako element potomny `ConstraintLayout`, na przykład a `Button`poniżej `TextView`, byłby on umieszczony za końcem `TextView`tagu `/>`i przed końcowym tagiem `ConstraintLayout`, w ten sposób:

```
<androidx.constraintlayout.widget.ConstraintLayout>
  <TextView
    android:text="Hello World!"/>
  <Button
    android:text="Calculate"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Więcej o XML dla układów

1. Spójrz na tag `ConstraintLayout`i zwróć uwagę, że `androidx.constraintlayout.widget.ConstraintLayout`zamiast tego jest napisany tak `ConstraintLayout`jak `TextView`. Dzieje się tak, ponieważ `ConstraintLayout`jest częścią Android Jetpack, który zawiera biblioteki kodu, które oferują dodatkowe funkcje poza podstawową platformą Android. Jetpack ma przydatną funkcjonalność, z której możesz skorzystać, aby ułatwić tworzenie aplikacji. Rozpoznasz ten składnik interfejsu użytkownika, który jest częścią Jetpack, ponieważ zaczyna się od „androidx”.
2. Być może zauważyłeś linie zaczynające się od `xmlns:`, po których następuje `android`, `app`i `tools`.

```
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"
```

`xmlns`Oznacza przestrzeń nazw XML, a każdy wiersz definiuje schemat *lub* słownik atrybutów powiązanych z tymi słowami. Na `android:`przykład przestrzeń nazw oznacza atrybuty zdefiniowane przez

system Android. Wszystkie atrybuty w układzie XML układu zaczynają się od jednej z tych przestrzeni nazw.

1. Białe znaki między elementami XML nie zmieniają znaczenia dla komputera, ale mogą ułatwić ludziom czytanie kodu XML.

Android Studio automatycznie doda białe znaki i wcięcia, aby zapewnić czytelność. Później dowiesz się, jak sprawić, by Android Studio upewniło się, że kod XML jest zgodny z konwencjami stylów kodowania.

1. Możesz dodawać komentarze do XML, tak jak w przypadku kodu Kotlin. Zacznij `<!--`i zakończ na `-->`.

```
<!-- thisis a comment in XML -->

<!-- thisis a
multi-line
Comment.
And another
Multi-linecomment -->
```

1. Zwróć uwagę na pierwszy wiersz pliku:

```
<?xml version="1.0"encoding="utf-8"?>
```

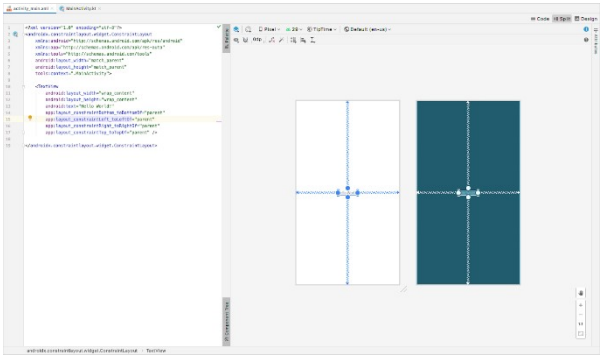
Oznacza to, że plik jest plikiem XML, ale nie każdy plik XML go zawiera.

Uwaga: jeśli wystąpi problem z kodem XML Twojej aplikacji, Android Studio oznaczy go, rysując tekst na czerwono. Jeśli najedziesz myszką na czerwony tekst, Android Studio wyświetli więcej informacji o problemie. Jeśli problem nie jest oczywisty, spójrz na wcięcia i kodowanie kolorami, a mogą one dać wskazówkę, co jest nie tak.

4. Zbuduj układ w XML

1. Będąc nadal w `activity_main.xml`trybie , przełącz się na widok **podzielonego** ekranu , aby zobaczyć kod XML obok **edytora projektu** . Edytor **projektu** umożliwia podgląd układu interfejsu użytkownika.

1. Wybór widoku zależy od osobistych preferencji, ale w tym ćwiczeniu z programowania użyj widoku **Split** , aby zobaczyć zarówno edytowany kod XML, jak i zmiany wprowadzane przez te edycje w **Edytorze projektu** .
2. Spróbuj kliknąć różne linie, jedną pod `ConstraintLayout`, a następnie pod , i zauważ, że w **Edytorze projektu**`TextView` wybrano odpowiedni widok . Odwrotność też działa — na przykład, jeśli klikniesz w **Edytorze projektu** , odpowiedni kod XML zostanie podświetlony.`TextView`



Usuń widok tekstu

1. Nie potrzebujesz `TextView`teraz, więc usuń to. Pamiętaj, aby usunąć wszystko od `<TextView`do zamknięcia `/>`.

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Hello World!"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintLeft_toLeftOf="parent"
   app:layout_constraintRight_toRightOf="parent"
   app:layout_constraintTop_toTopOf="parent"/>
```

Wszystko, co pozostało w pliku, to `ConstraintLayout`:

```
<androidx.constraintlayout.widget.ConstraintLayout
   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=".MainActivity">

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

1. Dodaj 16 dp dopełnienia, aby `ConstraintLayout`interfejs użytkownika nie był zatłoczony przy krawędzi ekranu.

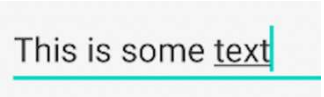
Dopełnienie jest podobne do marginesów, ale dodaje przestrzeń wewnątrz `ConstraintLayout`, zamiast dodawać przestrzeń na zewnątrz.

```
<androidx.constraintlayout.widget.ConstraintLayout
   ...
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:padding="16dp"
   tools:context=".MainActivity">
```

Uwaga: niektóre fragmenty kodu w tym ćwiczeniu z programowania nie pokażą wszystkiego dla zwięzłości. Kod, który się nie zmienił lub nie ma związku z bieżącym krokiem, będzie reprezentowany przez wielokropek (3 kolejne kropki ...), dzięki czemu możesz skupić się na najważniejszych częściach kodu.

Dodaj pole tekstowe kosztu usługi

W tym kroku dodasz element interfejsu użytkownika, aby umożliwić użytkownikowi wprowadzenie kosztu usługi w aplikacji. Użyjesz `EditText`elementu, który pozwala użytkownikowi wprowadzać lub modyfikować tekst w aplikacji.



1. Zapoznaj się z `EditText`dokumentacją i sprawdź przykładowy kod XML.
2. Znajdź pustą przestrzeń między otwierającym i zamykającym znacznikiem `ConstraintLayout`.
3. Skopiuj i wklej kod XML z dokumentacji do tego miejsca w układzie w Android Studio.

Twój plik układu powinien wyglądać tak:

```
<?xml version="1.0"encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
  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"
  android:padding="16dp"
  tools:context=".MainActivity">

  <EditText
    android:id="@+id/plain_text_input"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:inputType="text"/>

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

Możesz jeszcze tego wszystkiego nie zrozumieć, ale zostanie to wyjaśnione w kolejnych krokach.

1. Uwaga `EditText`jest podkreślona na czerwono.
2. Umieść wskaźnik myszy nad nim, a zobaczysz błąd „widok nie jest ograniczony”, który powinien wyglądać znajomo z wcześniejszych ćwiczeń z programowania. Przypomnij sobie, że dzieci `ConstraintLayout`potrzebują ograniczeń, aby układ wiedział, jak je rozmieścić.

1. Dodaj te ograniczenia do , EditTextaby zakotwiczyć go w lewym górnym rogu rodzica.

```
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
```

Jeśli piszesz po angielsku lub w innym języku pisanym od lewej do prawej (LTR), początkowa krawędź to lewa strona. Ale niektóre języki, takie jak arabski, są pisane od prawej do lewej (RTL), więc krawędź początkowa jest po prawej stronie. Dlatego ograniczenie używa "start", aby mogło działać z językami LTR lub RTL. Podobnie, ograniczenia używają słowa „koniec" zamiast prawej.

Uwaga: Nazwa ograniczeń ma postać layout_constraint<Source>_to<Target>Of, gdzie <Source>odnosi się do bieżącego widoku. <Target>odnosi się do krawędzi widoku docelowego, do którego ograniczony jest bieżący widok, albo do kontenera nadrzędnego, albo do innego widoku.

Po dodaniu nowych ograniczeń EditTextelement będzie wyglądał tak:

```
<EditText
  android:id="@+id/plain_text_input"
  android:layout_height="wrap_content"
  android:layout_width="match_parent"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toTopOf="parent"
  android:inputType="text"/>
```

Przejrzyj atrybuty EditText

Dokładnie sprawdź wszystkie EditTextwklejone atrybuty, aby upewnić się, że działa zgodnie z tym, jak będą używane w Twojej aplikacji.

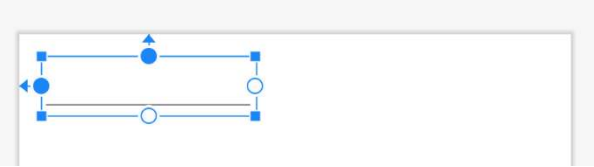
1. Znajdź idatrybut, który jest ustawiony na @+id/plain_text_input.
2. Zmień idatrybut na bardziej odpowiednią nazwę, @+id/cost_of_service.

Uwaga: Identyfikator zasobu to unikalna nazwa zasobu dla elementu. Gdy dodajesz Viewzasób lub inny zasób za pomocą **Edytora układu** , Android Studio automatycznie przypisuje im identyfikatory zasobów. Gdy ręcznie wpisujesz kod XML, musisz samodzielnie zadeklarować identyfikator zasobu. Nowe identyfikatory widoku w pliku XML muszą być zdefiniowane z @+idprefiksem, który informuje Android Studio o dodaniu tego identyfikatora jako nowego identyfikatora zasobu.

Wybierz opisowe nazwy zasobów, aby wiedzieć, do czego się odnoszą, ale wszystkie powinny być pisane małymi literami, a wiele słów powinno być oddzielonych podkreśleniem.

Odwołując się do identyfikatorów zasobów w kodzie aplikacji, użyj R.<type>.<name>; na przykład R.string.roll. W przypadku Viewidentyfikatorów <type>jest to idna przykład R.id.button.

1. Spójrz na layout_heightatrybut. Jest ustawiony na wrap_contentco oznacza, że jego wysokość będzie równa zawartości w środku. To jest w porządku, ponieważ będzie tylko 1 linia tekstu.
2. Spójrz na layout_widthatrybut. Jest ustawiony na match_parent, ale nie możesz ustawić match_parentna dziecko ConstraintLayout. Ponadto pole tekstowe nie musi być tak szerokie. Ustaw ją na stałą szerokość 160dp, która powinna zapewniać użytkownikowi dużo miejsca na wprowadzenie kosztu usługi.



1. Zwróć uwagę na `inputType`atrybut — to coś nowego. Wartość atrybutu to `"text"`, co oznacza, że użytkownik może wpisać w pole na ekranie dowolne znaki tekstowe (znaki alfabetyczne, symbole itp.)

```
android:inputType="text"
```

Jednak chcesz, aby w polu wpisywali tylko liczby `EditText`, ponieważ pole reprezentuje wartość pieniężną.

1. Usuń słowo `text`, ale pozostaw cudzysłowy.
2. Zacznij pisać `number`w jego miejscu. Po wpisaniu „n” Android Studio wyświetli listę możliwych uzupełnień zawierających „n”.

1. Wybierz `numberDecimal`, co ogranicza je do liczb z kropką dziesiętną.

```
android:inputType="numberDecimal"
```

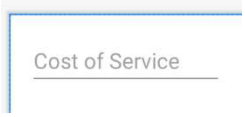
Aby zobaczyć inne opcje typów danych wejściowych, zobacz Określanie typu metody wprowadzania w dokumentacji dla programistów.

Do zrobienia jest jeszcze jedna zmiana, ponieważ warto wyświetlić wskazówkę, co użytkownik powinien wpisać w to pole.

1. Dodaj `hint`atrybut `EditText`opisujący, co użytkownik powinien wpisać w polu.

```
android:hint="Cost of Service"
```

Zobaczysz również aktualizację **Edytora projektu .**



1. Uruchom swoją aplikację w emulatorze. To powinno wyglądać tak:

Dobra robota! Niewiele jeszcze robi, ale masz dobry początek i edytowałeś trochę XML. XML układu powinien wyglądać mniej więcej tak.

```
<?xml version="1.0"encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
  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"
  android:padding="16dp"
  tools:context=".MainActivity">

  <EditText
    android:id="@+id/cost_of_service"
    android:layout_width="160dp"
    android:layout_height="wrap_content"
    android:hint="Cost of Service"
    android:inputType="numberDecimal"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

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

Dodaj pytanie serwisowe

W tym kroku dodasz `TextView`pytanie „Jak działa usługa?” Spróbuj wpisać to bez kopiowania/wklejania. Pomogą Ci sugestie Android Studio.

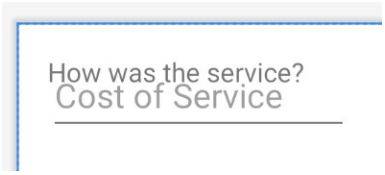
1. Po zamknięciu `EditText`tagu `/>`dodaj nową linię i zacznij pisać `<TextView`.
2. Wybierz jedną `TextView`z sugestii, a Android Studio automatycznie doda atrybuty `layout_width`i dla .`layout_height`TextView

3. Wybierz wrap_contentdla obu, ponieważ wystarczy, TextViewaby był tak duży, jak zawartość tekstu w środku.

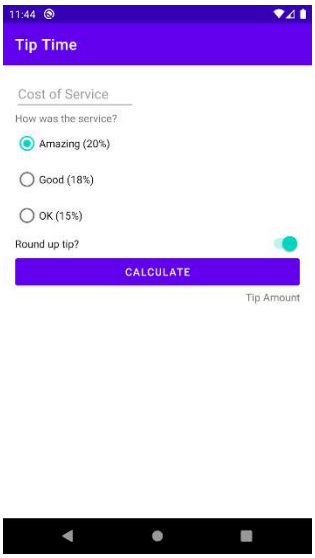
4. Dodaj textatrybut za pomocą"How was the service?"

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="How was the service?"
```

1. Zamknij tag za pomocą />.
2. Zwróć uwagę w **Edytorze projektu** , że TextViewzachodzą one na EditText.



To nie wygląda dobrze, więc dodasz ograniczenia w TextViewnastępnym. Zastanów się, jakich ograniczeń potrzebujesz. Gdzie należy TextViewustawić poziomo i pionowo? Możesz sprawdzić zrzut ekranu aplikacji, aby Ci pomóc.



Pionowo chcesz, TextViewaby pole tekstowe znajdowało się poniżej kosztu usługi. Poziomo, chcesz TextViewwyrównać do początkowej krawędzi rodzica.

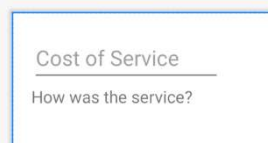
1. Dodaj wiązanie poziome do, TextViewaby powiązać jego początkową krawędź z początkową krawędzią rodzica.

```
app:layout_constraintStart_toStartOf="parent"
```

1. Dodaj wiązanie pionowe do , TextViewaby powiązać górną krawędź z TextViewdolną krawędzią kosztu usługi View.

app:layout_constraintTop_toBottomOf="@id/cost_of_service"

Zauważ, że nie ma plusa, `@id/cost_of_service`ponieważ identyfikator jest już zdefiniowany.



Nie wygląda to najlepiej, ale na razie nie martw się. Po prostu chcesz się upewnić, że wszystkie niezbędne elementy są na ekranie, a funkcjonalność działa. Poprawisz jego wygląd w poniższych ćwiczeniach z programowania.

1. Dodaj identyfikator zasobu na `TextView`. Będziesz musiał odwołać się do tego widoku później, gdy dodasz więcej widoków i powiążesz je ze sobą.

android:id="@+id/service_question"

W tym momencie twój XML powinien wyglądać tak.

```
<?xml version="1.0"encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
  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"
  android:padding="16dp"
  tools:context=".MainActivity">

  <EditText
    android:id="@+id/cost_of_service"
    android:hint="Cost of Service"
    android:layout_height="wrap_content"
    android:layout_width="160dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    android:inputType="numberDecimal"/>

  <TextView
    android:id="@+id/service_question"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="How was the service?"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/cost_of_service"/>

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

5. Dodaj opcje napiwków

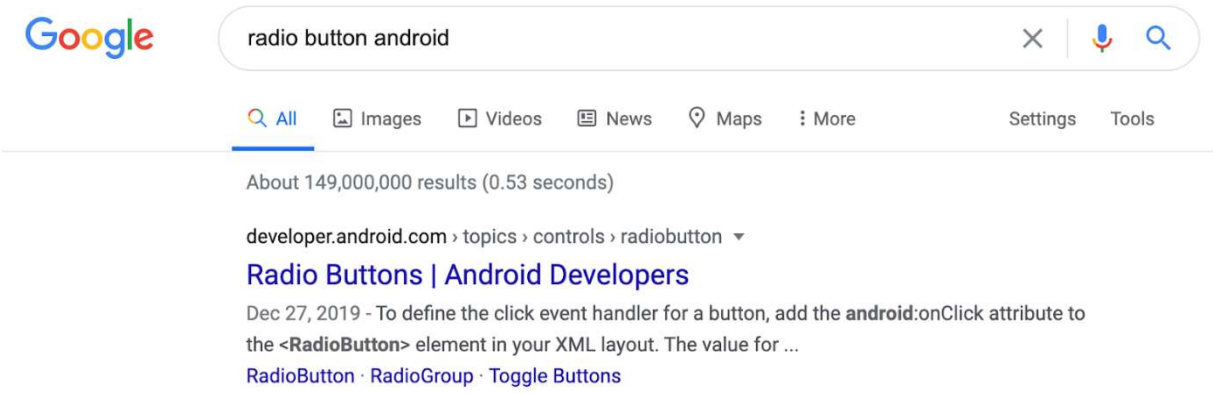
Następnie dodasz przyciski opcji dla różnych opcji wskazówek, z których użytkownik może wybierać.

Powinny istnieć trzy opcje:

- Niesamowite (20%)
- Dobry (18%)
- Dobrze (15%)

Jeśli nie wiesz, jak to zrobić, możesz przeprowadzić wyszukiwanie w Google. To świetne narzędzie, z którego programiści korzystają, gdy utkną.

1. Wyszukaj w Google `radio button android`. Najlepszym wynikiem jest przewodnik ze strony programistów Androida na temat korzystania z przycisków radiowych — idealnie!



1. Przejrzyj przewodnik po przyciskach radiowych .

Czytając opis, możesz potwierdzić, że możesz użyć `RadioButton`elementu interfejsu użytkownika w swoim układzie dla każdego potrzebnego przycisku opcji. Co więcej, musisz również pogrupować przyciski opcji w ramach a, `RadioGroup`ponieważ w danym momencie można wybrać tylko jedną opcję.

Istnieje kilka plików XML, które wyglądają, jakby pasowały do twoich potrzeb. Przeczytaj go i zobacz, jak `RadioGroup`wygląda widok nadrzędny i jakie `RadioButtons`są w nim widoki podrzędne.

1. Wróć do swojego układu w Android Studio, aby dodać `RadioGroup`i `RadioButton`do swojej aplikacji.
2. Po `TextView`elemencie, ale nadal w `ConstraintLayout`, zacznij pisać `<RadioGroup`. Android Studio zapewni przydatne sugestie, które pomogą Ci uzupełnić kod XML.

3. Ustaw `layout_width`i `layout_height`na . `RadioGroup_wrap_content`

4. Dodaj identyfikator zasobu ustawiony na `@+id/tip_options`.
5. Zamknij tag początkowy za pomocą `>`.
6. Android Studio dodaje `</RadioGroup>`. Podobnie jak element `ConstraintLayout`, `RadioGroup`element będzie zawierał inne elementy, więc możesz chcieć przenieść go do własnej linii.

7. Ogranicz `RadioGroup`poniższe pytanie serwisowe (w pionie) i do początku rodzica (w poziomie).
8. Ustaw `android:orientation`atrybut na `vertical`. Jeśli chcesz z `RadioButtons`rzędu, ustaw orientację na `horizontal`.

Kod XML `RadioGroup`powinien wyglądać tak:

```
<RadioGroup
  android:id="@+id/tip_options"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:orientation="vertical"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toBottomOf="@id/service_question">

</RadioGroup>
```

Dodaj przyciski radiowe

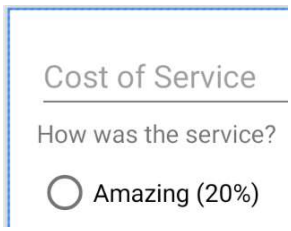
1. Po ostatnim atrybucie `RadioGroup`, ale przed `</RadioGroup>`znacznikiem końcowym dodaj `RadioButton`.

```
<RadioGroup
  android:id="@+id/tip_options"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:orientation="vertical"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toBottomOf="@id/service_question">

  <!-- addRadioButtonshere -->

</RadioGroup>
```

1. Ustaw `layout_width`i `layout_height`na `wrap_content`.

2. Przypisz identyfikator zasobu `@+id/option_twenty_percent`do `RadioButton`.
3. Ustaw tekst na `Amazing (20%)`.
4. Zamknij tag za pomocą `/>`.

```
<RadioGroup
  android:id="@+id/tip_options"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  app:layout_constraintTop_toBottomOf="@id/service_question"
  app:layout_constraintStart_toStartOf="parent"
  android:orientation="vertical">

  <RadioButton
    android:id="@+id/option_twenty_percent"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Amazing (20%)"/>

</RadioGroup>
```

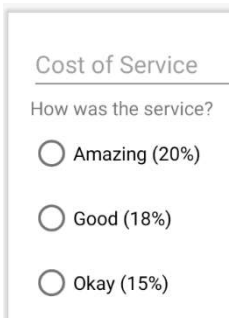


Teraz, po dodaniu jednego `RadioButton`, czy możesz zmodyfikować kod XML, aby dodać 2 dodatkowe przyciski opcji dla opcji `Good (18%)`i `Okay (15%)`?

Tak wygląda kod XML dla `RadioGroup`i `RadioButtons`:

```
<RadioGroup
  android:id="@+id/tip_options"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  app:layout_constraintTop_toBottomOf="@id/service_question"
  app:layout_constraintStart_toStartOf="parent"
  android:orientation="vertical">

  <RadioButton
    android:id="@+id/option_twenty_percent"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Amazing (20%)"/>

  <RadioButton
```

```
    android:id="@+id/option_eighteen_percent"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Good (18%)"/>

  <RadioButton
    android:id="@+id/option_fifteen_percent"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="OK (15%)"/>

</RadioGroup>
```



Dodaj domyślny wybór

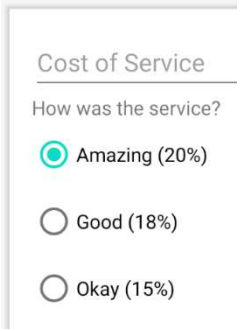
Obecnie żadna z opcji napiwków nie jest zaznaczona. Byłoby fajnie wybrać domyślnie jedną z opcji przycisków radiowych.

Istnieje atrybut, w `RadioGroup`którym możesz określić, który przycisk powinien być początkowo sprawdzany. Nazywa się `checkedButton`i ustawiasz go na identyfikator zasobu przycisku radiowego, który chcesz wybrać.

1. W dniu `RadioGroup`ustaw `android:checkedButton`atrybut na `@id/option_twenty_percent`.

```
<RadioGroup
  android:id="@+id/tip_options"
  android:checkedButton="@id/option_twenty_percent"
  ...
```

Zwróć uwagę w **Edytorze projektu** , że układ został zaktualizowany. Opcja 20% napiwku jest wybrana domyślnie — super! Teraz zaczyna wyglądać jak kalkulator napiwków!



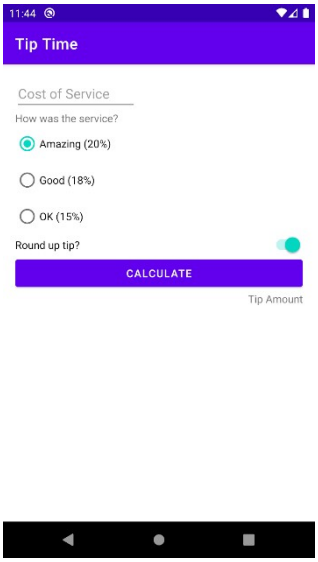
Oto jak do tej pory wygląda XML:

```
<?xml version="1.0"encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
  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"
  android:padding="16dp"
  tools:context=".MainActivity">

  <EditText
    android:id="@+id/cost_of_service"
    android:hint="Cost of Service"
    android:layout_height="wrap_content"
    android:layout_width="160dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    android:inputType="numberDecimal"/>

  <TextView
    android:id="@+id/service_question"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="How was the service?"
    app:layout_constraintTop_toBottomOf="@id/cost_of_service"
    app:layout_constraintStart_toStartOf="parent"/>

  <RadioGroup
    android:id="@+id/tip_options"
    android:checkedButton="@id/option_twenty_percent"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintTop_toBottomOf="@id/service_question"
    app:layout_constraintStart_toStartOf="parent"
    android:orientation="vertical">
```

```
    <RadioButton
      android:id="@+id/option_twenty_percent"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Amazing (20%)"/>

    <RadioButton
      android:id="@+id/option_eighteen_percent"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Good (18%)"/>

    <RadioButton
      android:id="@+id/option_fifteen_percent"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="OK (15%)"/>
  </RadioGroup>
</androidx.constraintlayout.widget.ConstraintLayout>
```

6. Uzupełnij resztę układu

Jesteś teraz na ostatniej części układu. Dodasz `Switch`, `Button`i a, `TextView`aby wyświetlić kwotę napiwku.



Dodaj przełącznik, aby zaokrąglić końcówkę

Następnie użyjesz `Switch`widżetu, aby umożliwić użytkownikowi wybranie tak lub nie w celu zaokrąglenia końcówki.

Chcesz `Switch`mieć taką samą szerokość jak rodzic, więc możesz pomyśleć, że szerokość powinna być ustawiona na `match_parent`. Jak wspomniano wcześniej, nie można ustawić `match_parent`elementów

interfejsu użytkownika w ConstraintLayout. Zamiast tego należy powiązać początkową i końcową krawędź widoku i ustawić szerokość na 0dp. Ustawienie szerokości na 0dpmówi systemowi, aby nie obliczał szerokości, po prostu spróbuj dopasować ograniczenia, które są na widoku.

Uwaga: nie możesz użyć match_parentdo żadnego widoku w ConstraintLayout. Zamiast tego użyj 0dp, co oznacza ograniczenia dopasowania.

1. Dodaj Switchelement po kodzie XML dla RadioGroup.
2. Jak wspomniano powyżej, ustaw layout_widthna 0dp.
3. Ustaw layout_heightna wrap_content. Dzięki temu Switchwidok będzie tak wysoki, jak zawartość w środku.
4. Ustaw idatrybut na @+id/round_up_switch.
5. Ustaw textatrybut na Rounduptip?. Będzie to używane jako etykieta dla Switch.
6. Powiąż początek krawędzi z Switchkrawędzią początkową tip_optionsi koniec z końcem rodzica.
7. Powiąż górę z Switchdolną częścią tip_options.
8. Zamknij tag za pomocą />.

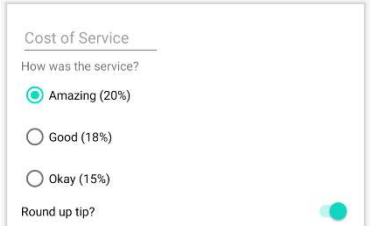
Byłoby fajnie, gdyby przełącznik był domyślnie włączony i jest dla niego atrybut android:checked, gdzie możliwe wartości to true(on) lub false(off).

1. Ustaw android:checkedatrybut na true.

Podsumowując, kod XML Switchelementu wygląda tak:

```
<Switch
  android:id="@+id/round_up_switch"
  android:layout_width="0dp"
  android:layout_height="wrap_content"
  android:checked="true"
  android:text="Rounduptip?"
  app:layout_constraintEnd_toEndOf="parent"
  app:layout_constraintStart_toStartOf="@id/tip_options"
  app:layout_constraintTop_toBottomOf="@id/tip_options"/>
```

Uwaga: jeśli najedziesz kursorem na Switchelement w układzie XML, zobaczysz sugestię w Android Studio, która mówi: „Użyj SwitchCompatzAppCompat lub SwitchMaterialz biblioteki Material”. Zaimplementujesz tę sugestię w późniejszym laboratorium programowania dotyczącym tworzenia aplikacji kalkulatora wskazówek, dzięki czemu możesz w międzyczasie zignorować ostrzeżenie.



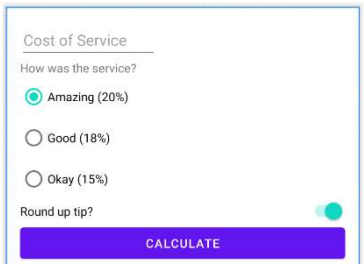
Dodaj przycisk Oblicz

Następnie dodasz znak Button, aby użytkownik mógł poprosić o obliczenie napiwku. Chcesz, aby przycisk był tak szeroki jak rodzic, więc poziome ograniczenia i szerokość są takie same, jak w przypadku Switch.

1. Dodaj a Buttonpo Switch.
2. Ustaw szerokość na 0dp, tak jak w przypadku Switch.
3. Ustaw wysokość na wrap_content.
4. Nadaj mu identyfikator zasobu @+id/calculate_button, z tekstem "Calculate".
5. Powiązać górną krawędź z Buttondolną krawędzią **końcówki Roundup?** Switch.
6. Powiąż krawędź początkową z początkową krawędzią rodzica, a końcową krawędź z końcową krawędzią rodzica.
7. Zamknij tag za pomocą />.

Oto jak wygląda kod XML dla **Calculate :**Button

```
<Button
  android:id="@+id/calculate_button"
  android:layout_width="0dp"
  android:layout_height="wrap_content"
  android:text="Calculate"
  app:layout_constraintTop_toBottomOf="@id/round_up_switch"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintEnd_toEndOf="parent"/>
```



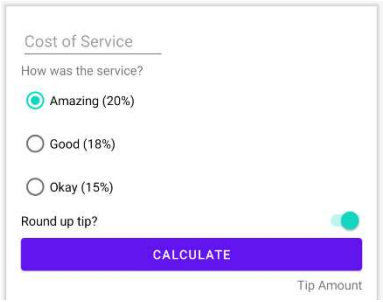
Dodaj wynik napiwku

Prawie skończyłeś z układem! W tym kroku dodasz TextViewwynik napiwku, umieszczając go poniżej przycisku **Oblicz** i wyrównaj z końcem zamiast z początkiem, tak jak inne elementy interfejsu użytkownika.

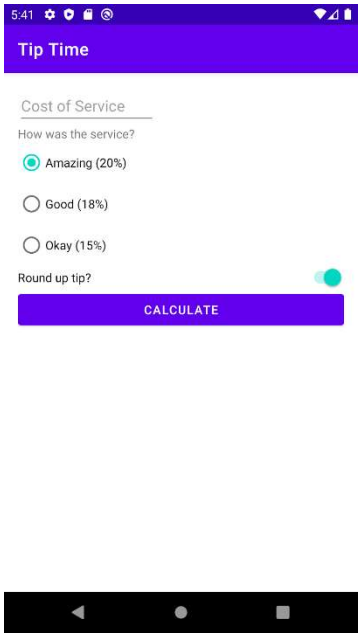
1. Dodaj a TextViewz nazwanym identyfikatorem zasobu tip_resulti tekstem TipAmount.
2. Powiąż końcową krawędź z TextViewkońcową krawędzią rodzica.
3. Powiąż górną krawędź z dolną krawędzią przycisku **Oblicz** .

```
<TextView
  android:id="@+id/tip_result"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  app:layout_constraintEnd_toEndOf="parent"
  app:layout_constraintTop_toBottomOf="@id/calculate_button"
```

android:text="TipAmount"/>



1. Uruchom aplikację. Powinien wyglądać jak ten zrzut ekranu.



Świetna robota, zwłaszcza jeśli po raz pierwszy pracujesz z XML!

Pamiętaj, że aplikacja może nie wyglądać dokładnie tak samo jak zrzut ekranu, ponieważ szablony mogły ulec zmianie w nowszej wersji Android Studio. Przycisk **Oblicz** nic jeszcze nie robi, ale możesz wpisać koszt, wybrać procent napiwku i przełączyć opcję zaokrąglania napiwku w górę lub nie. W następnym ćwiczeniu z programowania przycisk **Oblicz** będzie działał, więc nie zapomnij po niego wrócić!

7. Zastosuj dobre praktyki kodowania

Wyodrębnij struny

Być może zauważyłeś ostrzeżenia dotyczące ciągów zakodowanych na stałe. Przypomnij sobie z wcześniejszych ćwiczeń z programowania, że wyodrębnianie ciągów do pliku zasobów ułatwia tłumaczenie aplikacji na inne języki i ponowne używanie ciągów. Przejrzyj `activity_main.xml`i wyodrębnij wszystkie zasoby ciągów.

1. Kliknij ciąg; najedź na żółtą ikonę żarówki, która się pojawi, a następnie kliknij trójkąt obok niej; wybierz **Wyodrębnij zasób ciągu** . Domyślne nazwy zasobów tekstowych są w

porządku. Jeśli chcesz, jako opcje podpowiedzi możesz użyć `amazing_service`, `good_service`i , `okay_service`aby nazwy były bardziej opisowe.

Teraz zweryfikuj właśnie dodane zasoby tekstowe.

1. Jeśli okno **Projekt** nie jest wyświetlane, kliknij kartę **Projekt** po lewej stronie okna.
2. Otwórz **aplikację > res > wartości > strings.xml** , aby wyświetlić wszystkie zasoby ciągów interfejsu użytkownika.

```
<resources>
  <stringname="app_name">Tip Time</string>
  <stringname="cost_of_service">Cost of Service</string>
  <stringname="how_was_the_service">How was the service?</string>
  <stringname="amazing_service">Amazing (20%)</string>
  <stringname="good_service">Good (18%)</string>
  <stringname="ok_service">Okay (15%)</string>
  <stringname="round_up_tip">Rounduptip?</string>
  <stringname="calculate">Calculate</string>
  <stringname="tip_amount">TipAmount</string>
</resources>
```

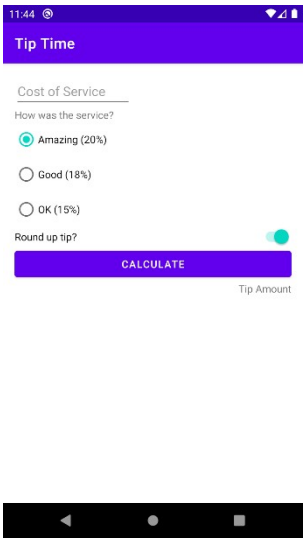
Sformatuj XML

Android Studio udostępnia różne narzędzia do uporządkowania kodu i upewnienia się, że jest on zgodny z zalecanymi konwencjami kodowania.

1. W programie `activity_main.xml`wybierz **Edycja > Zaznacz wszystko** .
2. Wybierz **Kod > Zmień format kodu** .

Zapewni to spójność wcięć i może zmienić kolejność niektórych elementów XML interfejsu użytkownika w celu pogrupowania elementów, na przykład połączenia wszystkich `android:`atrybutów jednego elementu.

8. Kod rozwiązania



res/layout/activity_main.xml

```
<?xml version="1.0"encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayoutxmlns: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"
   android:padding="16dp"
   tools:context=".MainActivity">

   <EditText
      android:id="@+id/cost_of_service"
      android:layout_width="160dp"
      android:layout_height="wrap_content"
      android:hint="@string/cost_of_service"
      android:inputType="numberDecimal"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent"/>

   <TextView
      android:id="@+id/service_question"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/how_was_the_service"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toBottomOf="@id/cost_of_service"/>

   <RadioGroup
      android:id="@+id/tip_options"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checkedButton="@id/option_twenty_percent"
    android:orientation="vertical"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/service_question">

    <RadioButton
        android:id="@+id/option_twenty_percent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/amazing_service"/>

    <RadioButton
        android:id="@+id/option_eighteen_percent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/good_service"/>

    <RadioButton
        android:id="@+id/option_fifteen_percent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ok_service"/>
</RadioGroup>

<Switch
    android:id="@+id/round_up_switch"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:checked="true"
    android:text="@string/round_up_tip"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="@id/tip_options"
    app:layout_constraintTop_toBottomOf="@id/tip_options"/>

<Button
    android:id="@+id/calculate_button"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="@string/calculate"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/round_up_switch"/>

<TextView
```

```
    android:id="@+id/tip_result"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/tip_amount"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@id/calculate_button"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

res/values/strings.xml

```
<resources>
  <stringname="app_name">Tip Time</string>
  <stringname="cost_of_service">Cost of Service</string>
  <stringname="how_was_the_service">How was the service?</string>
  <stringname="amazing_service">Amazing (20%)</string>
  <stringname="good_service">Good (18%)</string>
  <stringname="ok_service">Okay (15%)</string>
  <stringname="round_up_tip">Rounduptip?</string>
  <stringname="calculate">Calculate</string>
  <stringname="tip_amount">TipAmount</string>
</resources>
```

9. Podsumowanie

- XML (ExtensibleMarkup Language) to sposób organizowania tekstu, składający się ze znaczników, elementów i atrybutów.
- Użyj XML, aby zdefiniować układ aplikacji na Androida.
- Użyj `EditText`, aby umożliwić użytkownikowi wprowadzanie lub edycję tekstu.
- `EditText`Może mieć wskazówkę, aby powiedzieć użytkownikowi, czego oczekuje się w tym polu .
- Określ `android:inputType`atrybut, aby ograniczyć typ tekstu, który użytkownik może wprowadzić do `EditText`pola.
- Zrób listę ekskluzywnych opcji za pomocą `RadioButtons`, zgrupowanych za pomocą `RadioGroup`.
- A `RadioGroup`może być pionowe lub poziome i możesz określić, które `RadioButton`mają być wybrane na początku.
- Użyj a, `Switch`aby umożliwić użytkownikowi przełączanie między dwiema opcjami.
- Możesz dodać etykietę do `Switch`bez użycia oddzielnej `TextView`.
- Każde dziecko `ConstraintLayout`musi mieć ograniczenia w pionie i poziomie.
- Użyj ograniczeń „początek” i „koniec”, aby obsłużyć języki od lewej do prawej (LTR) i od prawej do lewej (RTL).
- Nazwy atrybutów ograniczeń są zgodne z formularzem `layout_constraint<Source>_to<Target>Of`.
- Aby szerokość była `View`tak szeroka, jak `ConstraintLayout`jest, ogranicz początek i koniec do początku i końca rodzica i ustaw szerokość na 0dp.

10. Dowiedz się więcej

Poniżej znajdują się łącza do dodatkowej dokumentacji na omawiane tematy. Całą dokumentację dotyczącą Android Development można znaleźć na stronie developer.android.com . I nie zapominaj, że możesz przeprowadzić wyszukiwanie w Google, jeśli coś utkniesz.

- TextView
- EditText
- Określ typ metody wprowadzania
- Przewodnik po przyciskach radiowych
- RadioButton
- RadioGroup
- Switch
- ConstraintLayout
- XML w Wikipedii

11. Ćwicz na własną rękę

Uwaga: Praktyki są opcjonalne. Dają ci możliwość przećwiczenia tego, czego nauczyłeś się podczas tego ćwiczenia z programowania.

Wykonaj następujące czynności:

- Utwórz inną aplikację kalkulatora, taką jak konwerter jednostek do gotowania, aby przekonwertować mililitry na lub z uncji płynu, gramy na lub z kubków i tak dalej. Jakich pól potrzebujesz?

Ścieżka 2. Uzyskaj dane wejściowe użytkownika w aplikacji: część 2

Dodaj wizualny połysk do aplikacji Kalkulator wskazówek, korzystając z wytycznych Material Design.

Zmień motyw aplikacji

1. Zanim zaczniesz

Material to system projektowania stworzony przez Google, aby pomóc programistom w tworzeniu wysokiej jakości cyfrowych doświadczeń na Androida i inne platformy. Pełny system Material zawiera wytyczne dotyczące projektowania wizualnego, ruchu i interakcji dla Twojej aplikacji, ale to laboratorium będzie koncentrować się na zmianie motywu kolorystycznego dla Twojej aplikacji na Androida. Ćwiczenia z programowania korzystają z szablonu aplikacji Empty Activity, ale możesz użyć dowolnej aplikacji na Androida, nad którą pracujesz. Jeśli bierzesz to jako część kursu Android Basics, możesz użyć aplikacji Tip Time .

Warunki wstępne

- Jak utworzyć aplikację na Androida z szablonu w Android Studio.
- Jak uruchomić aplikację na Androida na emulatorze lub urządzeniu w Android Studio.
- Urządzenie z systemem Android lub emulator z interfejsem API 28 (Android 9) lub API 29 (Android 10) lub nowszym.
- Jak edytować plik XML.

Czego się nauczysz

- Jak wybrać efektywne kolory dla swojej aplikacji zgodnie z zasadami Material Design
- Jak ustawić kolory jako część motywu aplikacji
- Składniki RGB koloru
- Jak zastosować styl doView
- Zmień wygląd aplikacji za pomocą motywu
- Zrozum znaczenie kontrastu kolorów

Czego potrzebujesz

- Komputer z zainstalowaną najnowszą stabilną wersją Android Studio
- Przeglądarka internetowa i połączenie internetowe umożliwiające dostęp do narzędzi koloru materiału

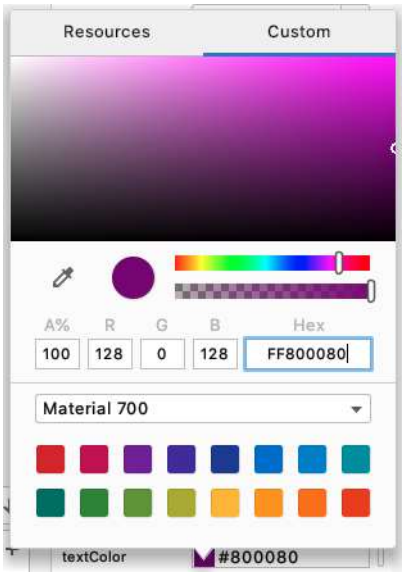
2. Projekt i kolor

Wygląd materiału

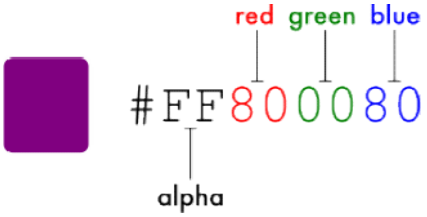
Material Design jest inspirowany światem fizycznym i jego teksturami, w tym sposobem, w jaki obiekty odbijają światło i rzucają cienie. Zawiera wytyczne dotyczące tworzenia interfejsu użytkownika aplikacji w czytelny, atrakcyjny i spójny sposób. Tematyka materiałów umożliwia dostosowanie Material Design do Twojej aplikacji, ze wskazówkami dotyczącymi dostosowywania kolorów, typografii i kształtów. Material Design zawiera wbudowany motyw bazowy, którego można używać bez zmian. Następnie możesz dostosować go tak mało lub tak bardzo, jak chcesz, aby materiał działał w Twojej aplikacji.

Trochę o kolorze

Kolor jest wokół nas, zarówno w świecie rzeczywistym, jak i cyfrowym. Pierwszą rzeczą, którą należy wiedzieć, jest to, że nie każdy widzi kolory w ten sam sposób, dlatego ważne jest, aby wybrać kolory dla aplikacji, aby użytkownicy mogli efektywnie z niej korzystać. Wybór kolorów o wystarczającym kontraście to tylko jeden z elementów tworzenia bardziej dostępnych aplikacji .



Kolor może być reprezentowany jako 3 liczby szesnastkowe, #00-#FF (0-255), reprezentujące składniki czerwony, zielony i niebieski (RGB) tego koloru. Im wyższa liczba, tym więcej tego składnika.



Zauważ, że kolor można również zdefiniować, włączając w to wartość alfa #00-#FF, która reprezentuje przezroczystość (#00 = 0% = całkowicie przezroczysty, #FF = 100% = całkowicie nieprzezroczysty). Jeśli jest uwzględniona, wartość alfa jest pierwszą z 4 liczb szesnastkowych (ARGB). Jeśli nie uwzględniono wartości alfa, przyjmuje się, że #FF = 100% nieprzezroczysty.

Nie musisz jednak samodzielnie generować liczb szesnastkowych. Dostępne są narzędzia, które pomogą Ci wybrać kolory, które wygenerują dla Ciebie liczby.

Niektóre przykłady, które mogłeś zobaczyć w `colors.xml`pliku aplikacji na Androida, obejmują 100% czerni (R=#00, G=#00, B=#00) i 100% bieli (R=#FF, G=#FF, B= #FF):

```
<color name="black">#FF000000</color>
<color name="white">#FFFFFFFF</color>
```

3. Motywy

Styl może określać atrybuty ,View takie jak kolor czcionki, rozmiar czcionki, kolor tła i wiele innych.

Motyw to zbiór stylów , które są stosowane do całej hierarchii aplikacji, działania lub widoków — nie tylko do poszczególnych osób View. Po zastosowaniu motywu do aplikacji, działania, widoku lub grupy widoków motyw jest stosowany do tego elementu i wszystkich jego elementów podrzędnych. Motywy mogą również stosować style do elementów innych niż widok, takich jak pasek stanu i tło okna.

Utwórz projekt pustej aktywności

Jeśli korzystasz z własnej aplikacji, możesz pominąć tę sekcję. Jeśli potrzebujesz aplikacji do pracy, wykonaj następujące kroki, aby utworzyć aplikację Pusta aktywność.

1. Otwórz Studio Androida.
2. Utwórz nowy projekt Kotlin, korzystając z szablonu **Empty Activity .**
3. Użyj nazwy „TipTime”. Możesz alternatywnie zachować domyślną nazwę „Moja aplikacja”, jeśli nie wykonujesz żadnych innych ćwiczeń z programowania.
4. Wybierz minimalny poziom interfejsu API 19 (KitKat).
5. Gdy Android Studio zakończy tworzenie aplikacji, otwórz activity_main.xml(**app > res > layout > activity_main.xml**).
6. W razie potrzeby przełącz się na widok **Kod** .
7. Zastąp cały tekst tym kodem XML:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
  android:layout_margin="16dp"
  android:orientation="vertical"
  tools:context=".MainActivity">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="48dp"
    android:layout_gravity="center_horizontal"
    android:gravity="center_vertical"
    android:text="@string/primary_color"
    android:textAllCaps="true"
    android:textSize="12sp" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
```

```
    android:text="@string/button" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="48dp"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="8dp"
    android:gravity="center_vertical"
    android:text="@string/secondary_color"
    android:textAllCaps="true"
    android:textSize="12sp" />

  <com.google.android.material.floatingactionbutton.FloatingActionButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:contentDescription="@string/email_icon"
    app:srcCompat="@android:drawable/ic_dialog_email" />

</LinearLayout>
```

8. Otwórz strings.xml(**app > res > wartości > strings.xml**).
9. Zastąp cały tekst tym kodem XML:

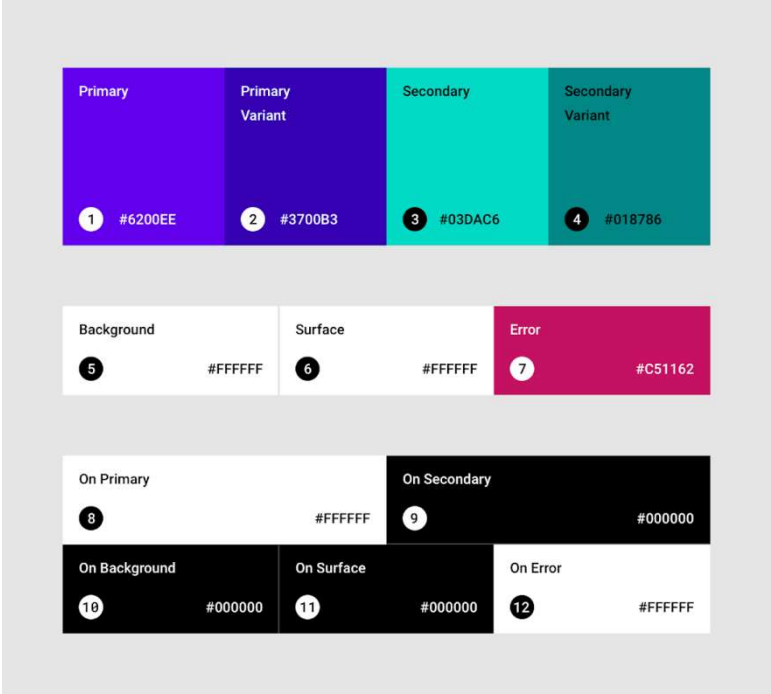
```
<resources>
  <string name="app_name">TipTime</string>
  <string name="primary_color">Primary color</string>
  <string name="button">Button</string>
  <string name="secondary_color">Secondary color</string>
  <string name="email_icon">email icon</string>
</resources>
```

10. Uruchom swoją aplikację. Aplikacja powinna wyglądać jak na poniższym zrzucie ekranu.

Aplikacja zawiera `TextView`i `Button`, aby zobaczyć, jak wyglądają wybrane przez Ciebie kolory w rzeczywistej aplikacji na Androida. W kolejnych krokach zmienimy kolor przycisku na kolor głównego koloru motywu.

Dowiedz się o kolorach motywu

Różne części interfejsu użytkownika aplikacji na Androida używają różnych kolorów. Aby ułatwić korzystanie z kolorów w aplikacji w znaczący sposób i konsekwentnie je stosować, system motywów grupuje kolory w 12 nazwanych atrybutów powiązanych z kolorami, które mają być używane przez tekst, ikony i nie tylko. Twój motyw nie musi określać ich wszystkich; będziesz wybierać kolory podstawowe i drugorzędne, a także kolory tekstu i ikon narysowanych na tych kolorach.



Kolory „Włączone” są używane dla tekstu i ikon rysowanych na różnych powierzchniach.

#	Nazwa	Atrybut motywu
1	Podstawowy	colorPrimary
2	Wariant podstawowy	colorPrimaryVariant
3	Wtórny	colorSecondary
4	Wariant drugorzędny	colorSecondaryVariant
5	Tło	colorBackground
6	Powierzchnia	colorSurface
7	Błąd	colorError
8	Na podstawowym	colorOnPrimary
9	Na drugorzędnym	colorOnSecondary
10	W tle	colorOnBackground
11	Na powierzchni	colorOnSurface
12	W przypadku błędu	colorOnError

Spójrz na kolory zdefiniowane w domyślnym motywie.

1. W Android Studio otwórz themes.xml(**app** > **res** > **wartości** > **motywy** > **motywy.xml**).
2. Zwróć uwagę na nazwę motywu Theme.TipTime, która jest oparta na nazwie Twojej aplikacji.

```
<style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
```

3. Zauważ, że wiersz XML określa również motyw nadrzędny, Theme.MaterialComponents.DayNight.DarkActionBar. DayNightjest predefiniowanym motywem w bibliotece Komponenty materiałów. DarkActionBaroznacza, że pasek akcji ma ciemny kolor. Tak jak klasa dziedziczy atrybuty z klasy nadrzędnej, motyw dziedziczy atrybuty z motywu nadrzędnego.

Uwaga: Atrybuty koloru motywu, które nie są zdefiniowane w motywie, będą używać koloru z motywu nadrzędnego.

4. Przejrzyj pozycje w pliku i zauważ, że nazwy są podobne do tych na powyższym diagramie: colorPrimary, colorSecondaryitd.

themes.xml

```
<resources xmlns:tools="http://schemas.android.com/tools">
  <!-- Base application theme. -->
  <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    <!-- Primary brand color. -->
    <item name="colorPrimary">@color/purple_500</item>
    <item name="colorPrimaryVariant">@color/purple_700</item>
```

```
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

Nie wszystkie atrybuty motywu kolorystycznego są zdefiniowane. Niezdefiniowane kolory odziedziczą kolor z motywu nadrzędnego.

5. Zauważ też, że Android Studio rysuje małą próbkę koloru na lewym marginesie.

```
4       <!-- Primary brand color. -->
5       <item name="colorPrimary">@color/purple_500</item>
6       <item name="colorPrimaryVariant">@color/purple_700</item>
7       <item name="colorOnPrimary">@color/white</item>
8       <!-- Secondary brand color. -->
9       <item name="colorSecondary">@color/teal_200</item>
10      <item name="colorSecondaryVariant">@color/teal_700</item>
11      <item name="colorOnSecondary">@color/black</item>
```

6. Na koniec zwróć uwagę, że kolory są określane jako zasoby kolorów, na przykład , `@color/purple_500`a nie bezpośrednio przy użyciu wartości RGB.
7. Otwórz `colors.xml`(**app** > **res** > **wartości** > **colors.xml**), a zobaczysz wartości szesnastkowe dla każdego zasobu koloru. Przypomnij sobie, że interlinia `#FF`jest wartością alfa i oznacza, że kolor jest w 100% nieprzezroczysty.

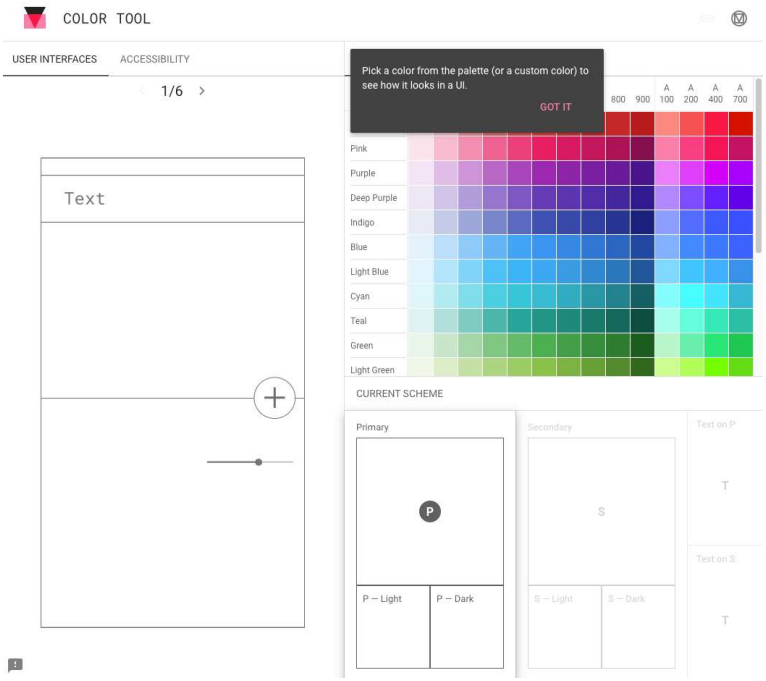
`colors.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```

4. Wybierz kolory motywu aplikacji

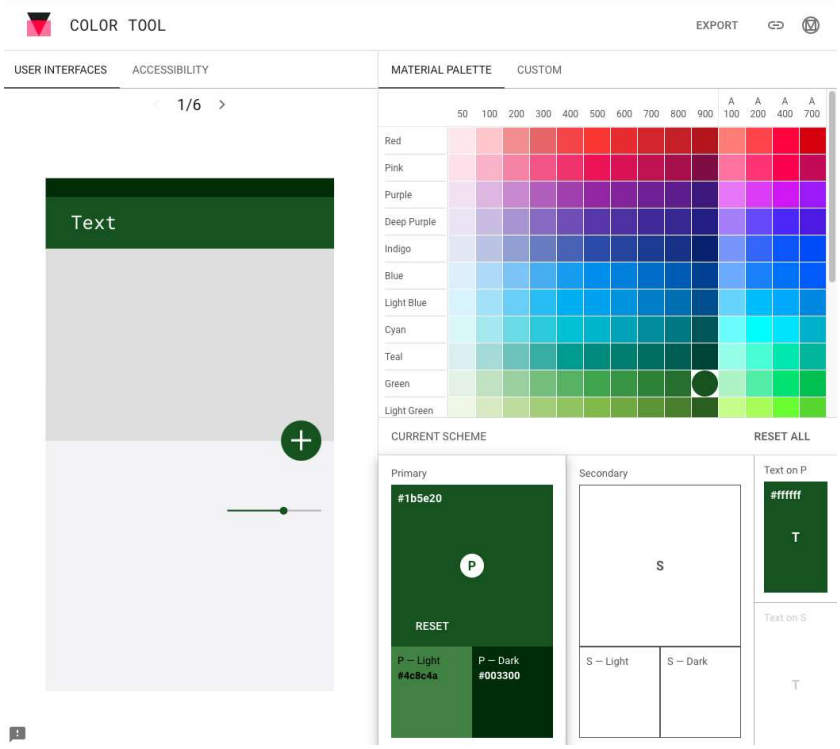
Teraz, gdy masz już pojęcie o atrybutach motywu, nadszedł czas, aby wybrać kilka kolorów! Najłatwiej to zrobić za pomocą narzędzia Color Tool , aplikacji internetowej udostępnionej przez zespół ds.

materiałów. Narzędzie udostępnia paletę wstępnie zdefiniowanych kolorów i pozwala łatwo zobaczyć, jak wyglądają, gdy są używane przez różne elementy interfejsu użytkownika.



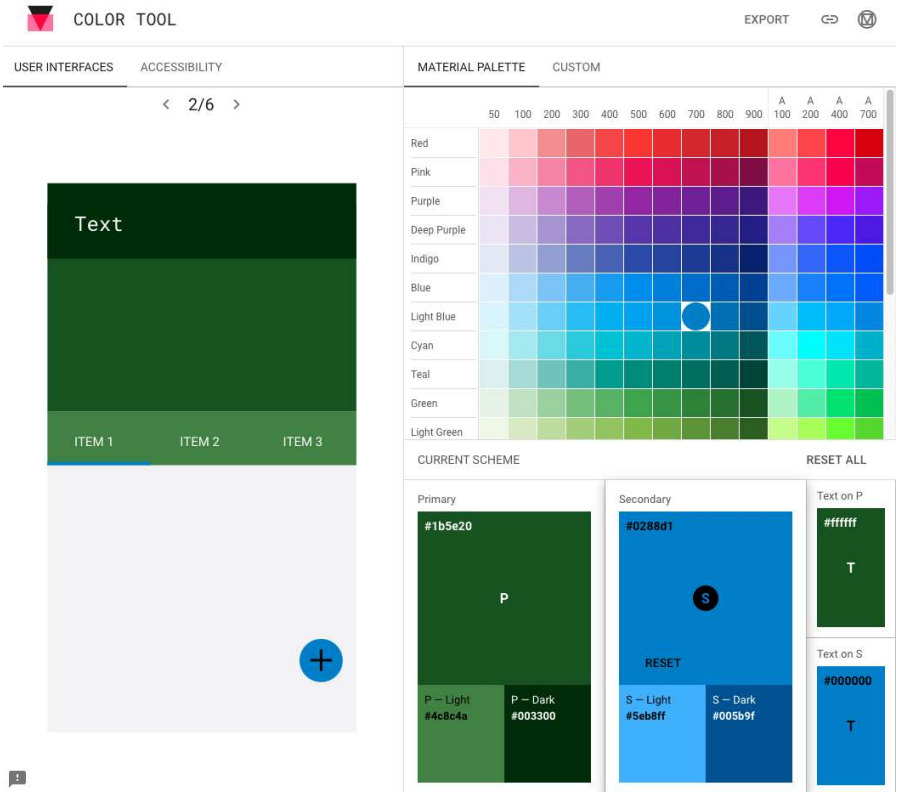
Wybierz kilka kolorów

1. Zacznij od wybrania koloru **podstawowego**colorPrimary ( ), na przykład **Zielony 900** . Narzędzie do kolorowania pokazuje, jak będzie wyglądać na makiecie aplikacji, a także wybiera warianty **jasne** i **ciemne**

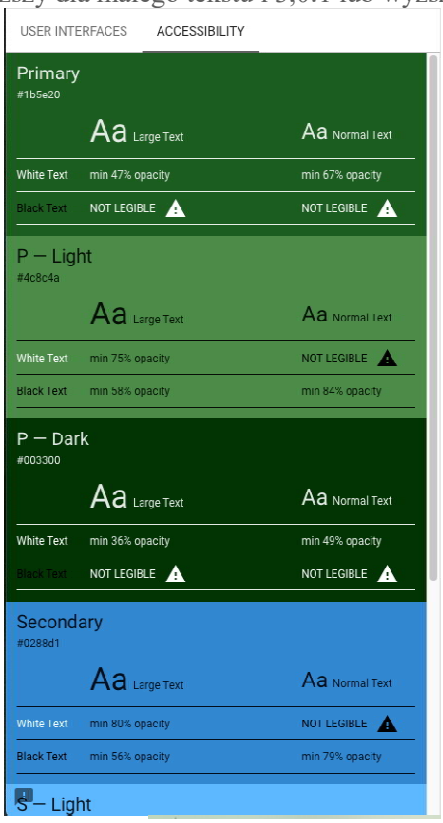


2. Stuknij w sekcję **Wtórny**colorSecondary i wybierz kolor dodatkowy ( ), który Ci się podoba, na przykład **Jasnoniebieski 700** . Kolor pokazuje, jak będzie wyglądać na makiecie aplikacji, i ponownie wybiera warianty **Jasny** i **Ciemny .**

3. Zwróć uwagę, że makieta aplikacji zawiera 6 różnych pozorowanych „ekranów”. Zobacz, jak wyglądają wybrane przez Ciebie kolory na różnych ekranach, dotykając strzałek nad makietą.



4. Narzędzie do kolorowania ma również kartę **Ułatwienia dostępu** , która informuje, czy kolory są wystarczająco wyraźne, aby można je było odczytać, gdy jest używane z czarnym lub białym tekstem. Częścią zwiększania dostępności aplikacji jest zapewnienie wystarczająco wysokiego kontrastu kolorów: 4,5:1 lub wyższy dla małego tekstu i 3,0:1 lub wyższy dla większego tekstu. Przeczytaj więcej o kontraście kolorów .



5. W przypadku primaryColorVarianti secondaryColorVariantmożesz wybrać sugerowany wariant jasny lub ciemny.

Uwaga: Możesz również użyć Generatora palety materiałów, aby wybrać kolor dodatkowy. Możesz wybrać kolor podstawowy, a on zasugeruje kolory uzupełniające , analogiczne lub triadyczne .

Dodaj kolory do swojej aplikacji

Zdefiniowanie zasobów dla kolorów ułatwia konsekwentne ponowne używanie tych samych kolorów w różnych częściach aplikacji.

1. W Android Studio otwórz colors.xml(**app > res > values > colors.xml**).
2. Po istniejących kolorach zdefiniuj zasób koloru o nazwie greenprzy użyciu wartości wybranej powyżej, #1B5E20.

```
<color name="green">#1B5E20</color>
```

3. Kontynuuj definiowanie zasobów dla innych kolorów. Większość z nich pochodzi z narzędzia do kolorowania. Zauważ, że wartości dla green_lighti blue_lightróżnią się od tego, co pokazuje narzędzie; użyjesz ich na późniejszym etapie.

green	#1B5E20
green_dark	#003300
green_light	#A5D6A7
blue	#0288D1
blue_dark	#005B9F
blue_light	#81D4FA

Istnieją już zasoby kolorów zdefiniowane dla czerni i bieli, więc nie musisz ich definiować.

Plik colors.xmlTwojej aplikacji powinien teraz wyglądać tak:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="purple_200">#FFBB86FC</color>
  <color name="purple_500">#FF6200EE</color>
  <color name="purple_700">#FF3700B3</color>
  <color name="teal_200">#FF03DAC5</color>
  <color name="teal_700">#FF018786</color>
  <color name="black">#FF000000</color>
  <color name="white">#FFFFFFFF</color>

  <color name="green">#1B5E20</color>
  <color name="green_dark">#003300</color>
  <color name="green_light">#A5D6A7</color>
  <color name="blue">#0288D1</color>
```

```
  <color name="blue_dark">#005B9F</color>
  <color name="blue_light">#81D4FA</color>
</resources>
```

Użyj kolorów w swoim motywie

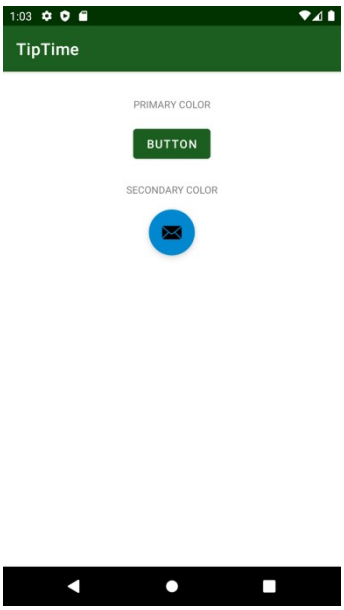
Teraz, gdy masz już nazwy dla wybranych kolorów, nadszedł czas, aby użyć ich w swoim motywie.

1. Otwórz `themes.xml`(**aplikacja** > **res** > **wartości** > **motywy** > **motywy.xml**).
2. Zmień `colorPrimary`na wybrany kolor podstawowy, `@color/green`.
3. Zmień `colorPrimaryVariant`na `@color/green_dark`.
4. Zmień `colorSecondary`na `@color/blue`.
5. Zmień `colorSecondaryVariant`na `@color/blue_dark`.
6. Sprawdź, czy **Tekst na P** i **Tekst na S** są nadal białe (`#FFFFFF`) i czarne (`#000000`). Jeśli korzystasz z narzędzia kolorów samodzielnie i wybierasz inne kolory, może być konieczne zdefiniowanie dodatkowych zasobów kolorów.

Kiedy skończysz, motyw powinien wyglądać tak:

```
<resources xmlns:tools="http://schemas.android.com/tools">
  <!-- Base application theme. -->
  <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    <!-- Primary brand color. -->
    <item name="colorPrimary">@color/green</item>
    <item name="colorPrimaryVariant">@color/green_dark</item>
    <item name="colorOnPrimary">@color/white</item>
    <!-- Secondary brand color. -->
    <item name="colorSecondary">@color/blue</item>
    <item name="colorSecondaryVariant">@color/blue_dark</item>
    <item name="colorOnSecondary">@color/black</item>
    <!-- Status bar color. -->
    <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
    <!-- Customize your theme here. -->
  </style>
</resources>
```

7. Uruchom swoją aplikację w emulatorze lub na urządzeniu i zobacz, jak wygląda Twoja aplikacja z nowym motywem.



5. Ciemny motyw

Szablon aplikacji zawierał domyślny jasny motyw, a także ciemny wariant motywu. Ciemny motyw wykorzystuje ciemniejsze, bardziej stonowane kolory oraz:

- Może znacznie zmniejszyć zużycie energii (w zależności od technologii ekranu urządzenia).
- Poprawia widoczność dla użytkowników słabo widzących i wrażliwych na jasne światło.
- Ułatwia każdemu korzystanie z urządzenia w warunkach słabego oświetlenia.

Wybór kolorów dla ciemnego motywu

Kolory ciemnego motywu nadal muszą być czytelne. Ciemne motywy wykorzystują ciemną powierzchnię z ograniczonymi akcentami kolorystycznymi. Aby zapewnić czytelność, kolory podstawowe są zwykle mniej nasyconymi wersjami kolorów podstawowych motywu jasnego.

Aby zapewnić większą elastyczność i użyteczność w ciemnym motywie, zaleca się użycie jaśniejszych tonów (200–50) w ciemnym motywie zamiast domyślnego motywu kolorów (nasycone tony w zakresie 900–500). Wcześniej wybrałeś zielony 200 i jasnoniebieski 200 jako jasne kolory. W swojej aplikacji użyjesz jasnych kolorów jako kolorów głównych, a kolorów podstawowych jako wariantów.

Zaktualizuj ciemną wersję swojego motywu

1. Otwórz `themes.xml (night)`**.** W okienku **projektu** wybierz system Android, przejdź do **aplikacji > res > wartości > motywy > themes.xml (noc)** .

   Uwaga: ten `themes.xml`plik różni się od poprzedniego `themes.xml`pliku. Ten plik zawiera ciemną wersję motywu. Zasoby w tym pliku będą używane, gdy na urządzeniu będzie włączony **ciemny motyw .**

2. Zmień `colorPrimary`na jasny wariant wybranego koloru podstawowego, `@color/green_light`.
3. Zmień `colorPrimaryVariant`na `@color/green`.
4. Zmień `colorSecondary`na `@color/blue_light`.

5. Zmień colorSecondaryVariantna @color/blue_light. Pamiętaj, że colorSecondaryVariantmoże to być ten sam kolor co colorSecondary.

   Kiedy skończysz, twój themes.xml (night)plik powinien wyglądać tak:

```
<resources xmlns:tools="http://schemas.android.com/tools">
  <!-- Application theme for dark theme. -->
  <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    <!-- Primary brand color. -->
    <item name="colorPrimary">@color/green_light</item>
    <item name="colorPrimaryVariant">@color/green</item>
    <item name="colorOnPrimary">@color/black</item>
    <!-- Secondary brand color. -->
    <item name="colorSecondary">@color/blue_light</item>
    <item name="colorSecondaryVariant">@color/blue_light</item>
    <item name="colorOnSecondary">@color/black</item>
    <!-- Status bar color. -->
    <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
    <!-- Customize your theme here. -->
  </style>
</resources>
```

6. W tym momencie oryginalne kolory zdefiniowane w colors.xml, na przykład , purple_200nie są już używane, więc można je usunąć.

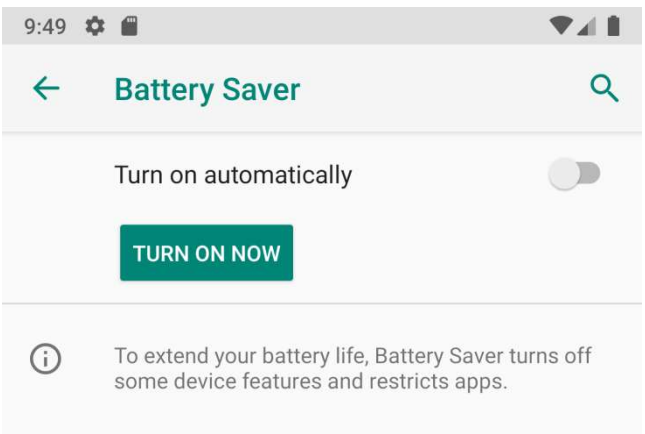
Wypróbuj ciemny motyw

Możesz zobaczyć, jak Twój motyw wygląda w trybie ciemnym, włączając tryb ciemny na swoim urządzeniu.

Uwaga: ciemny motyw wymaga urządzenia lub emulatora z interfejsem API 28 (Android 9) lub API 29 (Android 10) lub nowszym.

Dla API 28 (Android 9)

1. Uruchom ponownie swoją aplikację.
2. Przejdź do aplikacji **Ustawienia** .
3. W sekcji **Bateria** znajdź **Oszczędzanie baterii** .

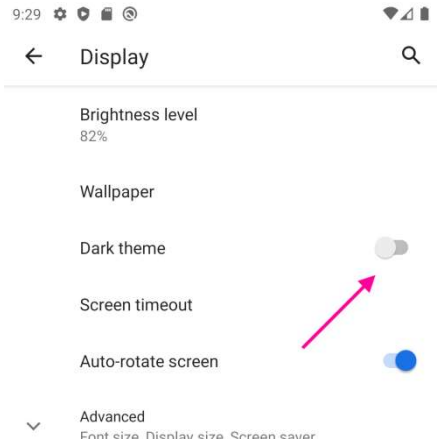


4. Naciśnij **Włącz teraz** .

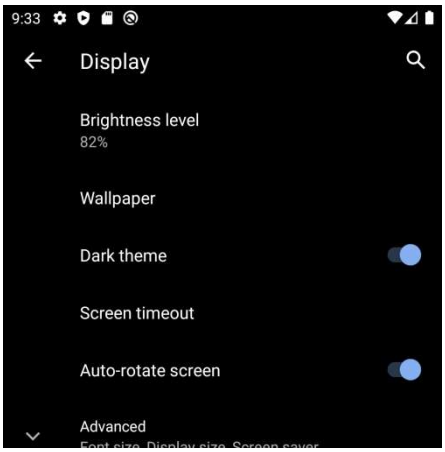
Kontynuuj, wykonując poniższe czynności.

Dla API 29 (Android 10) lub nowszego

1. Uruchom ponownie swoją aplikację.
2. Przejdź do aplikacji **Ustawienia** .
3. W sekcji **Wyświetlacz** znajdź przełącznik **Ciemny motyw** .

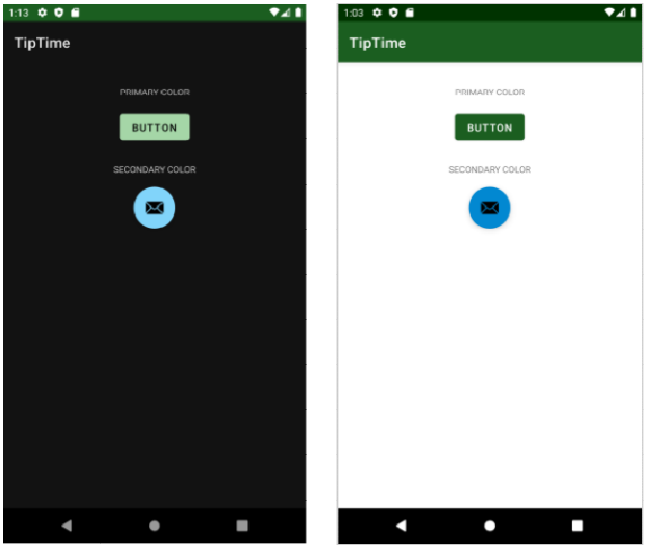


4. Przełącz **ciemny motyw** do pozycji „wł.”, a urządzenie przełączy się w tryb nocny.



Dla obu API

5. Wróć do swojej aplikacji i spójrz na różnice.



Najbardziej oczywistą zmianą jest ciemne tło z jasnym tekstem, zamiast jasnego tła z ciemnym tekstem. Ponadto kolory przycisków są mniej żywe w ciemnym motywie niż w jasnym motywie.

Gratulacje! Udało Ci się stworzyć nowy motyw aplikacji, zarówno z jasnym, jak i ciemnym motywem.

6. Kod rozwiązania

To laboratorium programowania koncentruje się na dostosowywaniu kolorów motywu, ale istnieje wiele atrybutów, które motyw może dostosować, w tym tekst, kształt, styl przycisku i inne. Zapoznaj się z tym artykułem, aby poznać inne sposoby dostosowywania motywu aplikacji! Stylizacja na Androida: wspólne atrybuty motywów .

Poniżej przedstawiono kod rozwiązania dla tego ćwiczenia z programowania.

colors.xml(**aplikacja > res > wartości > kolory.xml**)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="black">#FF000000</color>
   <color name="white">#FFFFFFFF</color>
   <color name="green">#1B5E20</color>
   <color name="green_dark">#003300</color>
   <color name="green_light">#A5D6A7</color>
   <color name="blue">#0288D1</color>
   <color name="blue_dark">#005B9F</color>
   <color name="blue_light">#81D4FA</color>
</resources>
```

values/themes.xml(**aplikacja > res > wartości > motywy > motywy.xml**)

```
<resources xmlns:tools="http://schemas.android.com/tools">
  <!-- Base application theme. -->
  <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    <!-- Primary brand color. -->
    <item name="colorPrimary">@color/green</item>
    <item name="colorPrimaryVariant">@color/green_dark</item>
    <item name="colorOnPrimary">@color/white</item>
    <!-- Secondary brand color. -->
    <item name="colorSecondary">@color/blue</item>
    <item name="colorSecondaryVariant">@color/blue_dark</item>
    <item name="colorOnSecondary">@color/black</item>
    <!-- Status bar color. -->
    <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
    <!-- Customize your theme here. -->
  </style>
</resources>
```

values-night/themes.xml (**aplikacja > res > wartości > motywy > motywy.xml (noc)**)

```
<resources xmlns:tools="http://schemas.android.com/tools">
  <!-- Application theme for dark theme. -->
  <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    <!-- Primary brand color. -->
    <item name="colorPrimary">@color/green_light</item>
    <item name="colorPrimaryVariant">@color/green</item>
    <item name="colorOnPrimary">@color/black</item>
    <!-- Secondary brand color. -->
    <item name="colorSecondary">@color/blue_light</item>
    <item name="colorSecondaryVariant">@color/blue_light</item>
    <item name="colorOnSecondary">@color/black</item>
    <!-- Status bar color. -->
    <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
    <!-- Customize your theme here. -->
  </style>
</resources>
```

7. Podsumowanie

- Użyj narzędzia koloru materiału , aby wybrać kolory dla motywu aplikacji.
- Alternatywnie możesz użyć Generatora palet materiałów, aby pomóc wybrać paletę kolorów.
- Zadeklaruj zasoby kolorów w colors.xmlpliku, aby ułatwić ich ponowne wykorzystanie.
- Ciemny motyw może zmniejszyć zużycie energii i ułatwić czytanie aplikacji przy słabym oświetleniu.

8. Dowiedz się więcej

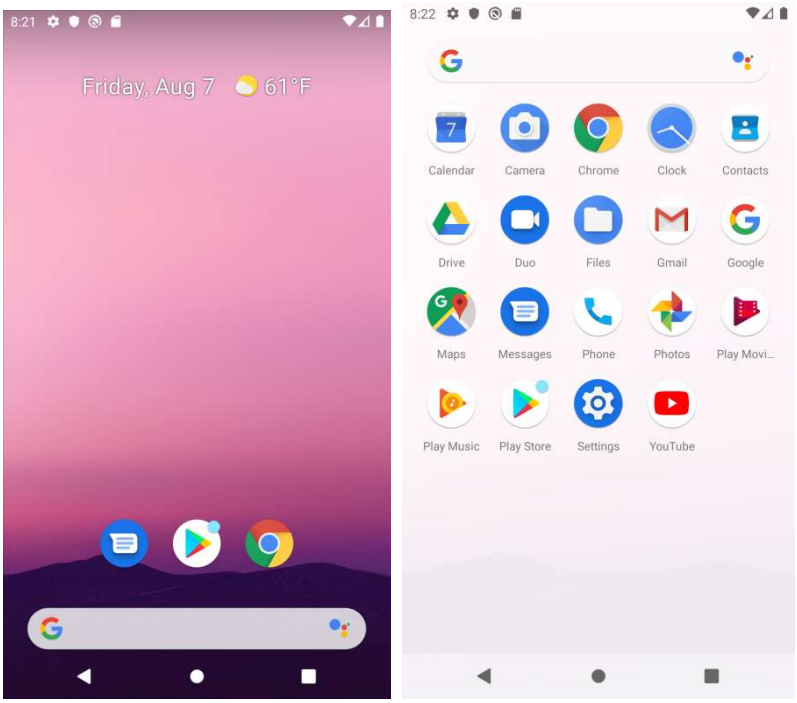
- System kolorów
- ciemny schemat
- Stylizacja na Androida: motywy a style
- Stylizacja na Androida: wspólne atrybuty motywów
- Konfigurowanie motywu komponentów materiałowych dla systemu Android
- Wskazówka dotycząca ciemnego motywu
- Kontroler kontrastu WebAIM
- Motywy komponentów materiałowych (patrz krok 4)
- Tematyka materiałów z MDC
- Ciemny motyw z MDC
- Stylizacja na Androida: wspólne atrybuty motywów

Zmień ikonę aplikacji

1. Zanim zaczniesz

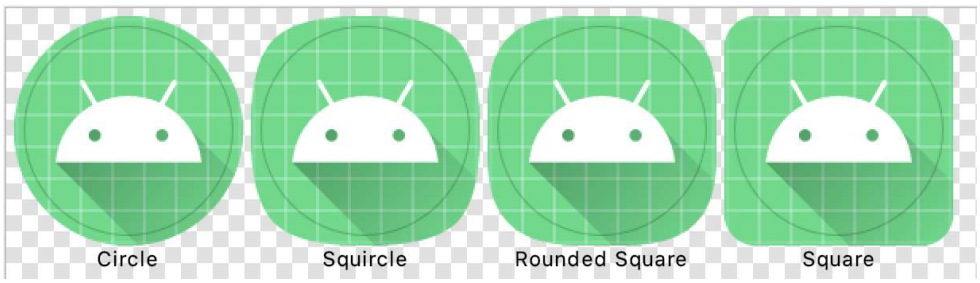
Ikona aplikacji to ważny sposób na wyróżnienie Twojej aplikacji. Pojawia się również w wielu miejscach, w tym na ekranie głównym, ekranie Wszystkie aplikacje i aplikacji Ustawienia.

Możesz również usłyszeć ikonę aplikacji, zwaną ikoną programu uruchamiającego. Launcher odnosi się do doświadczenia, gdy naciśniesz przycisk Home na urządzeniu z Androidem, aby wyświetlić i uporządkować swoje aplikacje, dodać widżety i skróty i nie tylko.



Jeśli wypróbowałeś różne urządzenia z Androidem, być może zauważyłeś, że środowisko Launchera może wyglądać inaczej w zależności od producenta urządzenia. Czasami producenci urządzeń tworzą niestandardowe środowisko Launchera, które jest znakiem rozpoznawczym ich marki. W związku z tym różni producenci mogą wyświetlać ikony aplikacji w innym kształcie niż okrągła ikona pokazana powyżej.

Mogą na przykład wyświetlać wszystkie ikony aplikacji w kształcie kwadratu, zaokrąglonego kwadratu lub squircle (między kwadratem a okręgiem).



Niezależnie od kształtu wybranego przez producentów urządzeń, celem jest, aby wszystkie ikony aplikacji na jednym urządzeniu miały jednolity kształt, aby zapewnić bardziej spójne wrażenia użytkownika.

Dlatego platforma Android wprowadziła obsługę ikon adaptacyjnych (od 26 poziomu API). Dzięki wdrożeniu adaptacyjnej ikony dla aplikacji Twoja aplikacja będzie mogła obsługiwać wiele różnych urządzeń dzięki odpowiedniemu wyświetlaniu wysokiej jakości ikony aplikacji.

W ramach tego ćwiczenia z kodowania otrzymasz pliki źródłowe obrazu dla ikony programu uruchamiającego Kalkulator wskazówek, z którymi możesz ćwiczyć. Użyjesz narzędzia w Android Studio o nazwie Image Asset Studio, aby wygenerować wszystkie potrzebne wersje ikon uruchamiania. Następnie możesz wziąć to, czego się nauczyłeś i zastosować to do zmiany ikony aplikacji dla innych aplikacji!

Warunki wstępne

- Potrafi nawigować po plikach podstawowego projektu Androida, w tym plikach zasobów
- Możliwość zainstalowania aplikacji na Androida z Android Studio na emulatorze lub urządzeniu fizycznym

Czego się nauczysz

- Jak zmienić ikonę programu uruchamiającego aplikacji?
- Jak używać Image Asset Studio w Android Studio do generowania zasobów ikon programu uruchamiającego
- Co to jest ikona adaptacyjna i dlaczego składa się z dwóch warstw

Co zbudujesz

- Aplikacja na Androida z nową ikoną programu uruchamiającego

Czego potrzebujesz

- Komputer z zainstalowaną najnowszą stabilną wersją Android Studio
- Połączenie internetowe w celu pobrania plików zasobów obrazu

2. Skonfiguruj swój projekt

Jeśli bierzesz udział w tym ćwiczeniu z kodowania w ramach kursu Android Basics in Kotlin , możesz bezpośrednio skorzystać z kalkulatora napiwków z poprzedniego ćwiczenia z kodowania, nad którym już pracujesz.

Jeśli wykonujesz samodzielną naukę programowania (poza kursem), możesz skonfigurować nowy projekt w Android Studio, korzystając z szablonu **Empty Activity .** W ten sposób nie będziesz modyfikować ani zastępować plików ikon programu uruchamiającego w istniejącej aplikacji, dopóki nie poczujesz się bardziej komfortowo z tymi krokami.

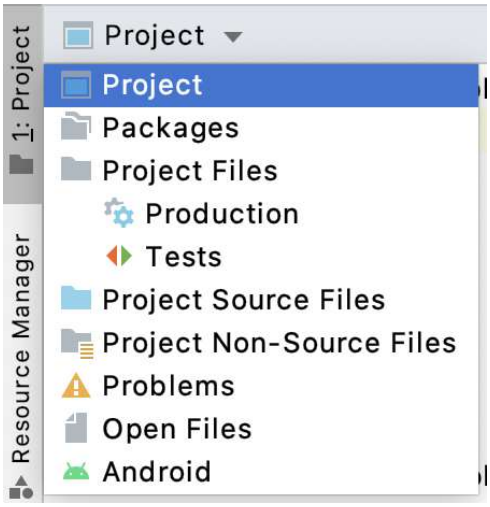
3. Ikony uruchamiania

Celem jest, aby ikona programu uruchamiającego wyglądała fantastycznie (ostry i wyraźny) niezależnie od modelu urządzenia lub gęstości ekranu. W szczególności gęstość pikseli ekranu odnosi się do liczby pikseli na cal (lub dpi, punktów na cal) na ekranie. W przypadku urządzenia o średniej gęstości (mdpi) na ekranie jest 160 punktów na cal, podczas gdy urządzenie o bardzo dużej gęstości (xxxhdpi) ma 640 punktów na cal na ekranie.

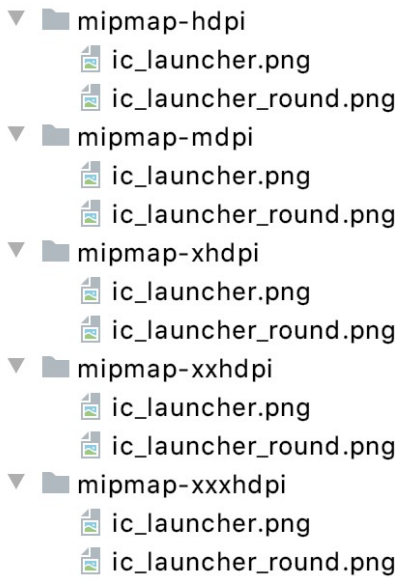
Aby uwzględnić urządzenia o różnych gęstościach ekranu, musisz podać różne wersje ikony aplikacji.

Przeglądaj pliki ikon programu uruchamiającego

1. Aby zobaczyć, jak to wygląda, otwórz swój projekt w Android Studio. Jeśli Twoja aplikacja została uruchomiona z szablonu, powinieneś mieć domyślne ikony uruchamiania, które są już dostarczane przez Android Studio.
2. W **oknie Projekt** przejdź do widoku **Projekt** . Spowoduje to wyświetlenie plików w twoim projekcie zgodnie ze strukturą plików, w jaki sposób te pliki są zapisywane na twoim komputerze.



3. Przejdź do katalogu zasobów (**app > src > main > res**) i rozwiń niektóre mipmapfoldery. W tych mipmapfolderach należy umieścić zasoby ikony programu uruchamiającego dla aplikacji na Androida.



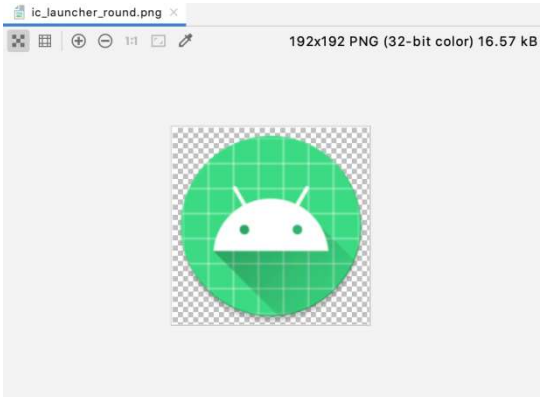
mdpi, hdpi, xhdpiitp. to kwalifikatory gęstości, które można dołączyć do nazwy katalogu zasobów (np mipmap.), aby wskazać, że są to zasoby dla urządzeń o określonej gęstości ekranu. Oto lista kwalifikatorów gęstości w systemie Android:

- mdpi- zasoby dla ekranów o średniej gęstości (~160 dpi)
- hdpi- zasoby dla ekranów o dużej gęstości (~240 dpi)
- xhdpi- zasoby dla ekranów o bardzo dużej gęstości (~320 dpi)
- xxhdpi- zasoby dla ekranów o bardzo dużej gęstości (~480 dpi)
- xxxhdpi- zasoby dla ekranów o bardzo dużej gęstości (~640 dpi)
- nodpi- zasoby, które nie są przeznaczone do skalowania, niezależnie od gęstości pikseli ekranu
- anydpi- zasoby skalowalne do dowolnej gęstości

Uwaga: możesz się zastanawiać, dlaczego zasoby ikon programu uruchamiającego znajdują się w mipmapkatalogach innych niż inne zasoby aplikacji znajdujące się w drawablekatalogach. Dzieje się tak, ponieważ niektóre programy uruchamiające mogą wyświetlać ikonę aplikacji w rozmiarze większym niż ten, który zapewnia domyślny zasobnik gęstości urządzenia. Na przykład na urządzeniu hdpi niektóre programy uruchamiające urządzenia mogą chcieć zamiast tego użyć wersji xhdpi ikony aplikacji.

4. Jeśli klikniesz na pliki graficzne, zobaczysz podgląd. Pliki ic_launcher.pngzawierają kwadratową wersję ikony, podczas gdy ic_launcher_round.pngpliki zawierają okrągłą wersję ikony. Oba są dostępne w każdym katalogu zasobów.

   Na przykład tak wygląda **res > mipmap-xxxhdpi > ic_launcher_round.png** . Zwróć też uwagę, że rozmiar zasobu znajduje się w prawym górnym rogu. Ten obraz ma rozmiar 192 pikseli x 192 pikseli.



Z drugiej strony tak wygląda **res > mipmap-mdpi > ic_launcher_round.png** . Ma tylko 48 x 48 pikseli.

Jak widać, te pliki obrazów bitmapowych składają się z ustalonej siatki pikseli. Zostały stworzone dla określonej rozdzielczości ekranu. W związku z tym jakość może ulec pogorszeniu podczas zmiany ich rozmiaru.

Jeśli zmniejszysz obraz bitmapowy, prawdopodobnie będzie wyglądał dobrze, ponieważ pozbywasz się informacji o pikselach. W przypadku znacznego skalowania obrazu bitmapowego może on wyglądać niewyraźnie, ponieważ system Android będzie musiał odgadnąć i uzupełnić brakujące informacje o pikselach.

Uwaga : aby uniknąć rozmycia ikony aplikacji, należy zapewnić różne obrazy bitmapowe ikony aplikacji dla każdego segmentu gęstości (mdpi, hdpi, xhdpiitp.). Należy pamiętać, że gęstość ekranu urządzenia nie będzie dokładnie wynosić 160 dpi, 240 dpi, 320 dpi itp. ... Na podstawie aktualnej gęstości ekranu system Android wybierze zasób w najbliższym zasobniku o większej gęstości, a następnie zmniejszy skalę.

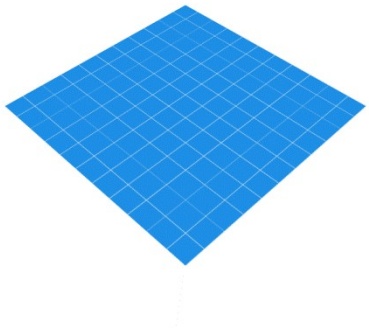
Teraz, gdy masz już trochę kontekstu tła na ikonach programu uruchamiającego, w dalszej części dowiesz się o ikonach adaptacyjnych.

4. Ikony adaptacyjne

Warstwy pierwszego planu i tła

Począwszy od wydania Androida 8.0 (poziom API 26), istnieje obsługa adaptacyjnych ikon uruchamiania, co pozwala na większą elastyczność i ciekawe efekty wizualne, jeśli chodzi o ikony aplikacji. Dla

programistów oznacza to, że ikona Twojej aplikacji składa się z dwóch warstw: pierwszego planu i warstwy tła.

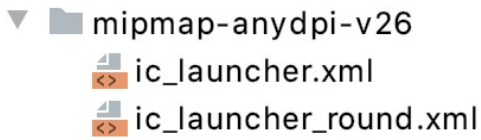


W powyższym przykładzie biała ikona Androida znajduje się na warstwie pierwszego planu, a niebieska i biała siatka na warstwie tła. Warstwa pierwszego planu zostanie ułożona na warstwie tła. Następnie maska (w tym przypadku okrągła) zostanie zastosowana na górze, aby utworzyć okrągłą ikonę aplikacji.

Przeglądaj adaptacyjne pliki ikon

Spójrz na domyślne pliki ikon adaptacyjnych dostarczone przez szablon projektu w Android Studio.

1. W **oknie projektu** Android Studio poszukaj i rozwiń katalog zasobów **res > mipmap-anydpi-v26** .

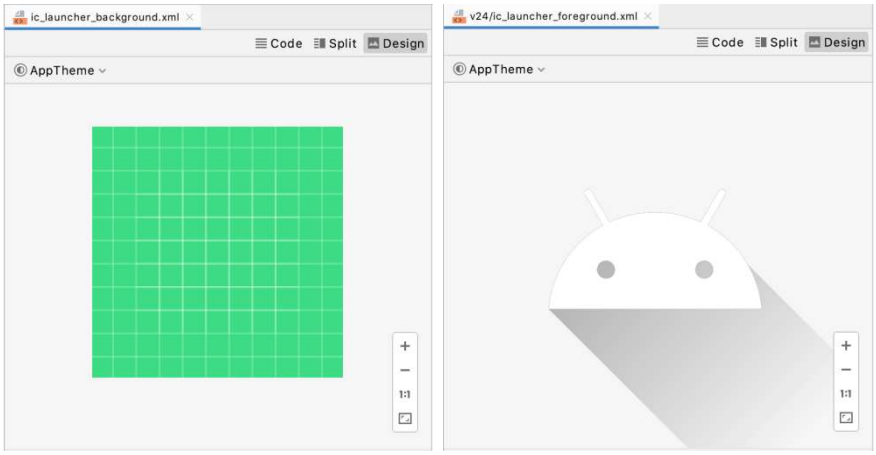


Uwaga : Ikony adaptacyjne zostały dodane w API level 26 platformy, więc ikony adaptacyjne powinny być zadeklarowane w `mipmap`katalogu zasobów, w którym znajduje się `-v26`kwalifikator zasobu. Oznacza to, że zasoby w tym katalogu będą stosowane tylko na urządzeniach z interfejsem API 26 (Android 8.0) lub nowszym. Pliki zasobów w tym katalogu są ignorowane na urządzeniach ze starszymi wersjami platformy.

2. Otwórz jeden z plików XML, na przykład `ic_launcher.xml`. Powinieneś to zobaczyć:

```
<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">
  <background android:drawable="@drawable/ic_launcher_background" />
  <foreground android:drawable="@drawable/ic_launcher_foreground" />
</adaptive-icon>
```

3. Zwróć uwagę, w jaki sposób `<adaptive-icon>`element jest używany do deklarowania warstw `<background>`i `<foreground>`ikony aplikacji, dostarczając dla każdej z nich elementy do rysowania zasobów.
4. Wróć do widoku **projektu** i poszukaj, gdzie są zadeklarowane obiekty do rysowania: **drawable > ic_launcher_background.xml** i **drawable-v24 > ic_launcher_foreground.xml** .
5. Przełącz się do widoku **Projekt** , aby zobaczyć podgląd każdego z nich (tło po lewej, pierwszy plan po prawej).



6. Oba są plikami wektorowymi do rysowania. Nie mają stałego rozmiaru w pikselach. Jeśli przełączysz się do widoku **Kod** , możesz zobaczyć deklarację XML dla wektora, który można narysować za pomocą <vector>elementu.

   Chociaż zarówno obraz wektorowy, jak i obraz bitmapowy opisują grafikę, istnieją ważne różnice.

   Obraz bitmapowy niewiele rozumie o przechowywanym obrazie, z wyjątkiem informacji o kolorze w każdym pikselu. Z drugiej strony grafika wektorowa wie, jak narysować kształty, które definiują obraz. Instrukcje te składają się z zestawu punktów, linii i krzywych wraz z informacjami o kolorze. Zaletą jest to, że grafikę wektorową można skalować do dowolnego rozmiaru płótna przy dowolnej gęstości ekranu, bez utraty jakości.

   Wektor **do rysowania** to implementacja grafiki wektorowej w systemie Android, która ma być wystarczająco elastyczna na urządzeniach mobilnych. Możesz je zdefiniować w XML z tymi możliwymi elementami . Zamiast udostępniać wersje zasobu mapy bitowej dla wszystkich zasobników gęstości, wystarczy zdefiniować obraz tylko raz. W ten sposób zmniejszając rozmiar aplikacji i ułatwiając jej utrzymanie.
   Uwaga: istnieją kompromisy między używaniem do rysowania wektorowego a obrazem bitmapowym. Na przykład ikony mogą być idealne do rysowania wektorów, ponieważ składają się z prostych kształtów, podczas gdy fotografię trudniej byłoby opisać jako serię kształtów. W takim przypadku bardziej efektywne byłoby użycie zasobu mapy bitowej.

   Teraz nadszedł czas, aby przejść do faktycznej zmiany ikony aplikacji!

5. Pobierz nowe zasoby

Następnie pobierz te 2 nowe zasoby, które pozwolą Ci utworzyć adaptacyjną ikonę dla aplikacji Kalkulator napiwków. Nie musisz się martwić o zrozumienie każdego szczegółu plików wektorowych do rysowania. Ich zawartość może zostać automatycznie wygenerowana z narzędzi do projektowania.

1. Pobierz ic_launcher_background.xml, wektor do rysowania dla warstwy tła. Jeśli przeglądarka wyświetla plik zamiast go pobierać, wybierz **Plik > Zapisz stronę jako...** , aby zapisać go na komputerze.
2. Pobierz ic_launcher_foreground.xml, wektor do rysowania dla warstwy pierwszego planu.
   Należy pamiętać, że istnieją pewne wymagania dotyczące tych zasobów warstwy pierwszego planu i tła, na przykład oba muszą mieć rozmiar 108 dp x 108 dp. Więcej szczegółów na temat wymagań można znaleźć tutaj lub zapoznać się ze wskazówkami projektowymi dotyczącymi ikon Androida na stronie Materiał.

Ponieważ krawędzie Twojej ikony mogą zostać przycięte w zależności od kształtu maski producenta urządzenia, ważne jest, aby kluczowe informacje o Twojej ikonie umieścić w „ bezpiecznej strefie ”, czyli okręgu o średnicy 66 dp w środku warstwa. Treść poza tą bezpieczną strefą nie powinna być istotna (np. kolor tła), gdzie można ją przyciąć.

6. Zmień ikonę aplikacji

Wróć do Android Studio, aby korzystać z nowych zasobów.

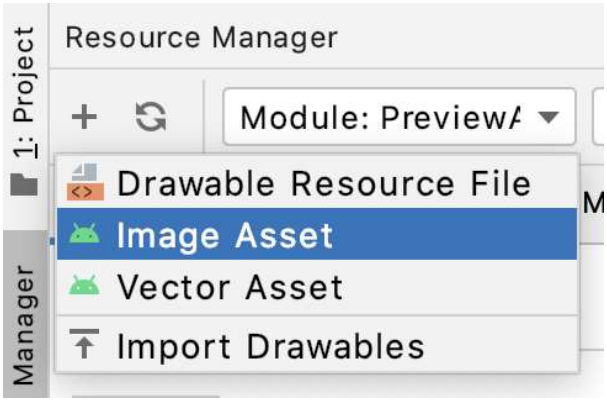
1. Najpierw usuń stare zasoby do rysowania, które mają ikonę Androida i zielone tło siatki. W **widoku projektu** kliknij prawym przyciskiem myszy plik i wybierz **Usuń** .

   Usuwać:

   drawable/ic_launcher_background.xml
   drawable-v24/ic_launcher_foreground.xml

   Możesz odznaczyć pole **Bezpieczne usuwanie (z wyszukiwaniem użycia)** i kliknąć **OK** .

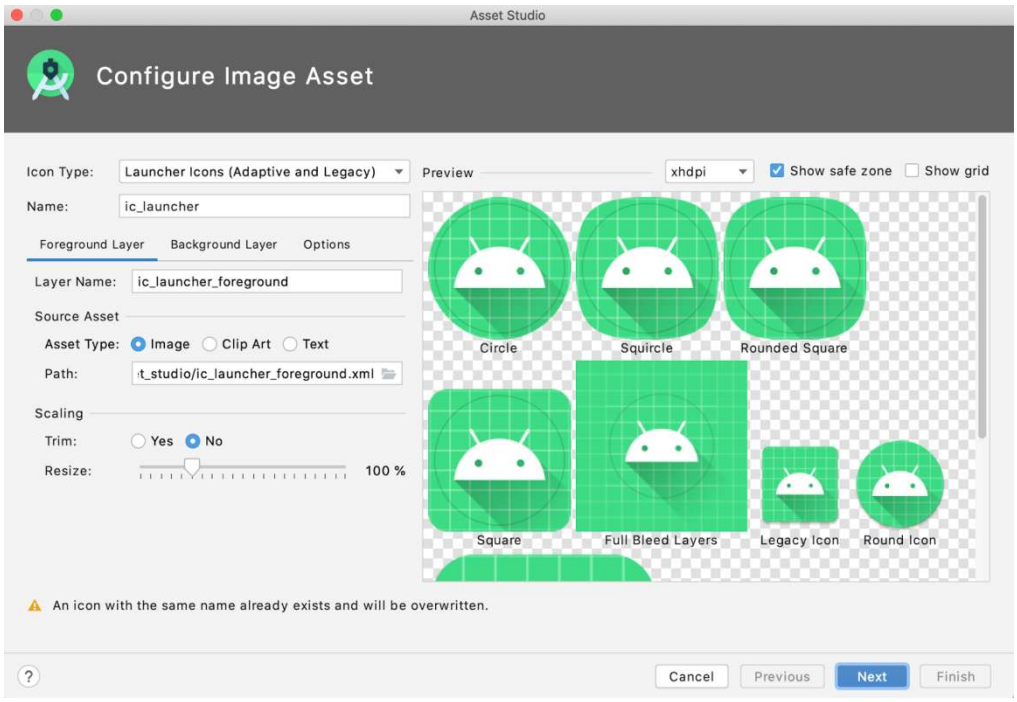
2. Utwórz nowy **zasób graficzny** . Możesz kliknąć prawym przyciskiem myszy katalog **res i wybrać Nowy> Zasób obrazu.** Możesz też kliknąć kartę **Menedżer zasobów** , kliknąć ikonę + i wybrać **Zasób obrazu.**



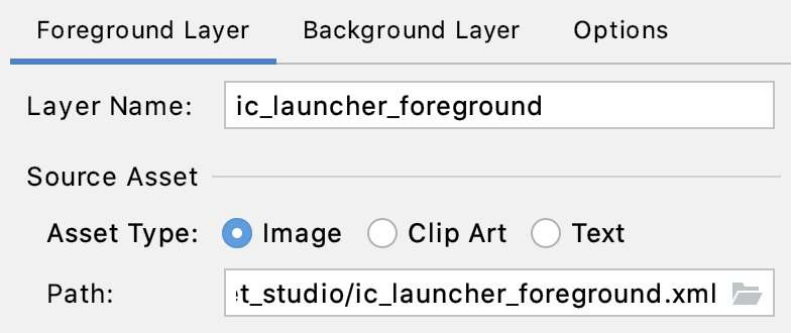
3. Otworzy się narzędzie **Image Asset Studio** w Android Studio .
4. Pozostaw ustawienia domyślne:

   Typ ikony: Ikony programu uruchamiającego (adaptacyjne i starsze)

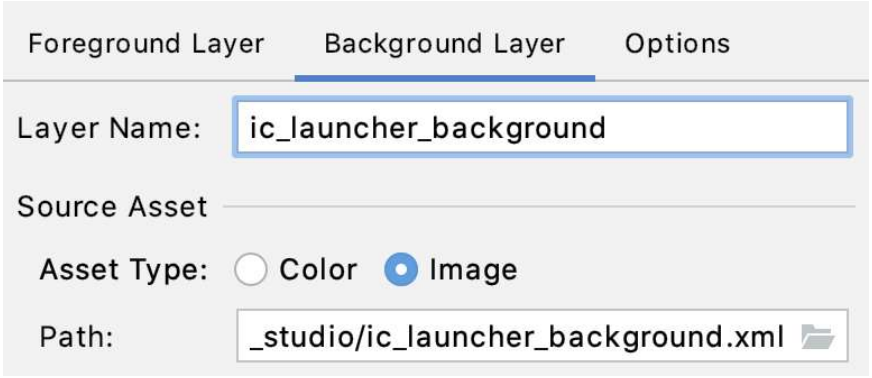
   Nazwa: ic_launcher



5. Po wybraniu **karty Warstwy pierwszego planu** przejdź do podsekcji **Zasób źródłowy .** W polu **Ścieżka** kliknij ikonę folderu.
6. Pojawi się monit, aby przejrzeć komputer i wybrać plik. Znajdź lokalizację nowego `ic_launcher_foreground.xml`pliku, który właśnie pobrałeś na swój komputer. Może znajdować się w folderze pobierania na twoim komputerze. Po znalezieniu kliknij **Otwórz** .
7. Ścieżka jest teraz zaktualizowana o położenie nowego do **rysowania** wektora pierwszego planu. Pozostaw **nazwę warstwy** jako **ic_launcher_foreground** i **typ zasobu** jako **obraz** .



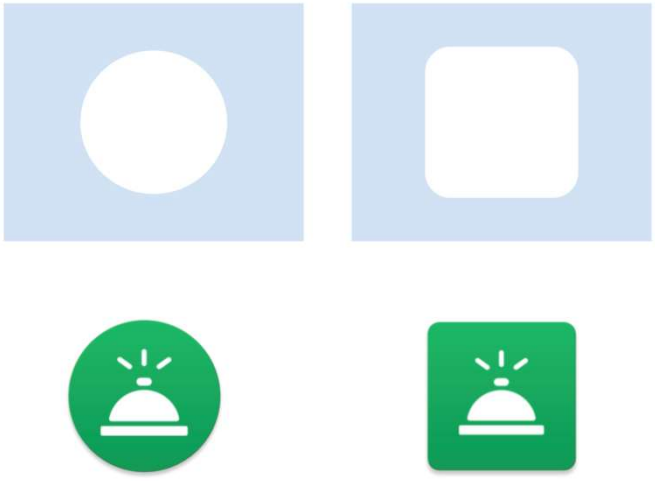
8. Następnie przejdź do **zakładki Warstwa tła** interfejsu. Pozostaw wartości domyślne bez zmian. Kliknij ikonę folderu **Ścieżki** .
9. Znajdź lokalizację `ic_launcher_background.xml`właśnie pobranego pliku. Kliknij **Otwórz** .



10. Podgląd powinien być aktualizowany w miarę wybierania nowych plików zasobów. Tak powinien wyglądać z nowymi warstwami pierwszego planu i tła.

Reprezentując ikonę aplikacji na dwóch warstwach, producenci urządzeń (nazywani w skrócie producentami oryginalnego sprzętu lub OEM) mogą tworzyć różne kształty w zależności od urządzenia z systemem Android, jak pokazano na powyższym podglądzie. Producent OEM zapewnia maskę, która jest stosowana do wszystkich ikon aplikacji na urządzeniu.

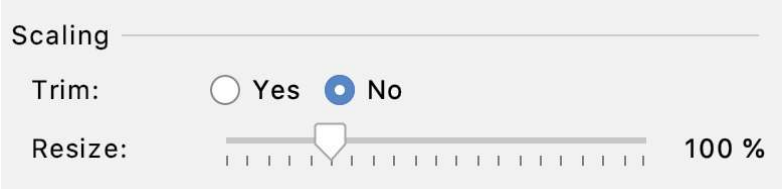
Ta maska jest nakładana na warstwy pierwszego planu i tła ikony aplikacji. Poniżej przykład maski okrągłej i maski kwadratowej.



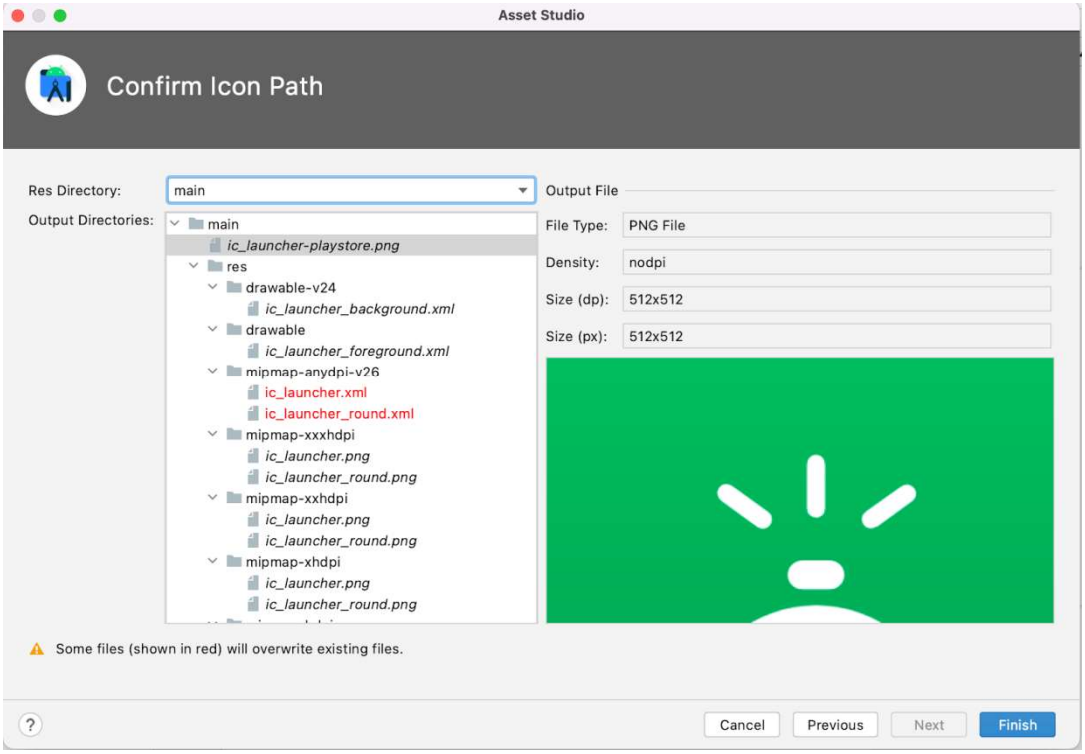
Gdy okrągła maska zostanie zastosowana do obu warstw ikony aplikacji, w wyniku pojawi się okrągła ikona z niebieskim tłem siatki i systemem Android (lewy obrazek powyżej). Alternatywnie możesz zastosować kwadratową maskę, aby wyświetlić ikonę aplikacji po prawej stronie powyżej.

Posiadanie dwóch warstw pozwala również na uzyskanie ciekawych efektów wizualnych, ponieważ dwie warstwy mogą poruszać się niezależnie lub być skalowane. Aby poznać zabawne przykłady tego, jak mogą wyglądać efekty wizualne, zapoznaj się z tym wpisem na blogu w części Zagadnienia projektowe. Ponieważ nie możesz z góry powiedzieć, jakie urządzenie będzie miał Twój użytkownik ani jaką maskę OEM zastosuje do Twojej ikony, musisz skonfigurować ikonę adaptacyjną, aby ważne informacje nie zostały odcięte.

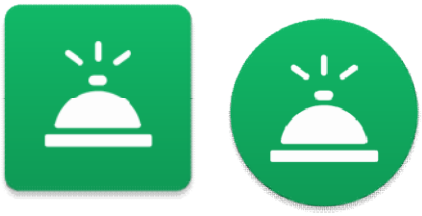
11. Upewnij się, że główna zawartość warstwy pierwszego planu (w tym przypadku ikona dzwonka usługi) znajduje się w bezpiecznej strefie i nie jest przycięta przez różne kształty masek. Jeśli ważna treść jest przycięta lub wydaje się zbyt mała, możesz użyć suwaka **Zmień rozmiar w sekcji Skalowanie** każdej warstwy. W takim przypadku nie jest wymagana zmiana rozmiaru, więc możesz pozostawić to na 100%.



12. Kliknij **Dalej** .
13. Ten krok polega na **potwierdzeniu ścieżki ikony** . Możesz kliknąć poszczególne pliki, aby zobaczyć podgląd. Na dole znajduje się również ostrzeżenie, że niektóre istniejące pliki zostaną nadpisane (pokazane na czerwono). To jest w porządku, ponieważ te stare pliki dotyczyły poprzedniej ikony aplikacji.



14. Domyślne ustawienia są w porządku, więc kliknij **Zakończ** .
15. Sprawdź, czy wszystkie wygenerowane zasoby wyglądają poprawnie w `mipmap`folderach. Przykłady:



Świetna robota! Teraz dokonasz jeszcze jednej zmiany.

Przenieś wektory do rysowania do katalogu -v26

W zależności od minimalnego pakietu SDK aplikacji możesz zauważyć, że zasób tła znajduje się w `drawable-v24`folderze, a zasób pierwszego planu znajduje się w `drawable`folderze. Powodem jest to, że zasób tła zawiera gradient, który był dostępny od wersji Androida 7.0 (znanej również jako wersja API 24, stąd `-v24`kwalifikator zasobów). Zasób pierwszego planu nie zawiera gradientu, więc można go umieścić w `drawable`folderze podstawowym.

Zamiast umieszczać zasoby pierwszego planu i tła w dwóch oddzielnych `drawable`folderach, przenieś oba pliki wektorowe do rysowania do `-v26`katalogu zasobów. Ponieważ te zasoby są używane tylko do ikon adaptacyjnych, te dwa elementy do rysowania są potrzebne tylko w interfejsie API 26 i nowszych. Taka struktura folderów ułatwi znajdowanie plików ikon adaptacyjnych i zarządzanie nimi.

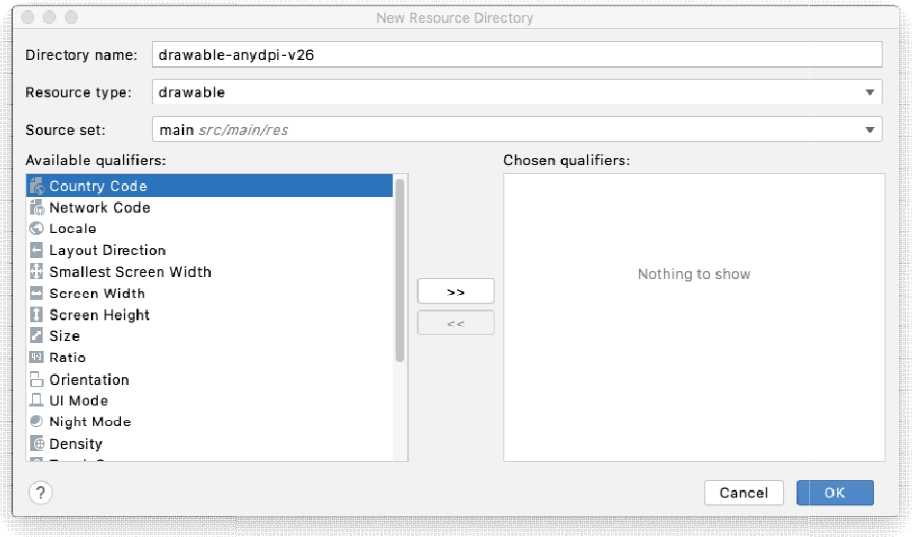
```
drawable-anydpi-v26
  ic_launcher_background.xml
  ic_launcher_foreground.xml
mipmap-anydpi-v26
  ic_launcher.xml
  ic_launcher_round.xml
```

1. Najpierw utwórz `drawable-anydpi-v26`katalog. Kliknij prawym przyciskiem myszy folder **res .** Wybierz pozycję **Nowy > Katalog zasobów systemu Android.**
2. Pojawi się okno dialogowe **nowego katalogu zasobów .** Wybierz te opcje:

   Nazwa katalogu : drawable-anydpi-v26

   Typ zasobu: do rysowania (wybierz z listy rozwijanej)

   Zestaw źródłowy: główny (pozostaw wartość domyślną bez zmian)

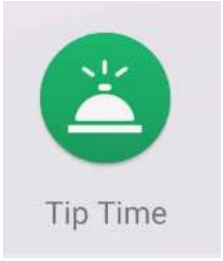


Kliknij **OK** . W widoku **projektu** sprawdź, czy został utworzony nowy katalog zasobów **res > drawable-anydpi-v26** .

3. Kliknij plik lewym przyciskiem myszy ic_launcher_foreground.xmli przeciągnij go z folderu do rysowania do folderu **drawable -anydpi-v26** . Przypomnijmy, że umieszczenie zasobu w katalogu „dowolne dpi" wskazuje, że jest to zasób, który można skalować do dowolnej gęstości.
4. Kliknij lewym przyciskiem myszy ic_launcher_background.xmli przeciągnij go z folderu **drawable-v24** do folderu **drawable-anydpi-v26** .
5. Usuń drawable-v24folder, jeśli jest teraz pusty. Kliknij prawym przyciskiem myszy folder, a następnie wybierz **Usuń** .
6. Kliknij wszystkie pliki drawablei w swoim projekcie. mipmapUpewnij się, że podgląd tych ikon wygląda poprawnie.

Przetestuj swoją aplikację

1. Sprawdź, czy pojawiła się nowa ikona aplikacji. Uruchom aplikację na swoim urządzeniu (emulatorze lub urządzeniu fizycznym).
2. Naciśnij przycisk home na swoim urządzeniu.
3. Przesuń w górę, aby wyświetlić listę Wszystkie aplikacje.
4. Poszukaj właśnie zaktualizowanej aplikacji. Powinieneś zobaczyć nową ikonę aplikacji.



Uwaga: w zależności od modelu urządzenia możesz zobaczyć ikonę programu uruchamiającego o innym kształcie. Niemniej jednak powinien pokazywać warstwę pierwszego planu na warstwie tła z nałożoną na nią maską.

Dobra robota! Nowa ikona aplikacji wygląda świetnie.

Adaptacyjne i starsze ikony programu uruchamiającego

Teraz, gdy Twoja ikona adaptacyjna działa dobrze, możesz się zastanawiać, dlaczego nie możesz pozbyć się wszystkich obrazów bitmapowych ikon aplikacji. Nadal potrzebujesz tych plików, aby ikona Twojej aplikacji była wyświetlana w wysokiej jakości w starszych wersjach Androida, co jest określane jako kompatybilność wsteczna.

Na urządzeniach z systemem Android 8.0 lub nowszym (API w wersji 26 lub nowszej):

Można używać ikon adaptacyjnych (kombinacja rysowania wektora pierwszego planu, rysowania wektora tła, z nałożoną maską OEM). Oto odpowiednie pliki w Twoim projekcie:

res/drawable-anydpi-v26/ic_launcher_background.xml
res/drawable-anydpi-v26/ic_launcher_foreground.xml
res/mipmap-anydpi-v26/ic_launcher.xml
res/mipmap-anydpi-v26/ic_launcher_round.xml

Na urządzeniach z systemem niższym niż Android 8.0 (ale powyżej minimalnego wymaganego poziomu API naszej aplikacji):

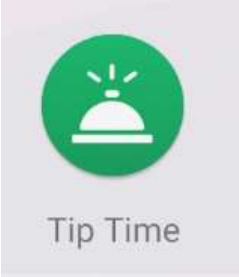
Zostaną użyte **starsze ikony programu uruchamiającego**mipmap (obrazy bitmapowe w folderach zasobników o różnej gęstości). Oto odpowiednie pliki w Twoim projekcie:

res/mipmap-mdpi/ic_launcher.png
res/mipmap-mdpi/ic_launcher_round.png
res/mipmap-hdpi/ic_launcher.png
res/mipmap-hdpi/ic_launcher_round.png
res/mipmap-xhdpi/ic_launcher.png
res/mipmap-xhdpi/ic_launcher_round.png
res/mipmap-xxdpi/ic_launcher.png
res/mipmap-xxdpi/ic_launcher_round.png
res/mipmap-xxxdpi/ic_launcher.png
res/mipmap-xxxdpi/ic_launcher_round.png

Zasadniczo Android powróci do obrazów bitmapowych na starszych urządzeniach bez adaptacyjnej obsługi ikon.

Gratulacje, wykonałeś wszystkie kroki zmiany ikony aplikacji!

7. Kod rozwiązania



Poniżej znajduje się kod rozwiązania dla tego ćwiczenia z programowania.

res/mipmap-anydpi-v26/ic_launcher.xml

```
<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">
   <background android:drawable="@drawable/ic_launcher_background" />
   <foreground android:drawable="@drawable/ic_launcher_foreground" />
</adaptive-icon>
```

res/mipmap-anydpi-v26/ic_launcher_round.xml

```
<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">
   <background android:drawable="@drawable/ic_launcher_background" />
   <foreground android:drawable="@drawable/ic_launcher_foreground" />
</adaptive-icon>
```

res/drawable-anydpi-v26/ic_launcher_background.xml

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:aapt="http://schemas.android.com/aapt"
   android:width="108dp"
   android:height="108dp"
   android:viewportWidth="108"
   android:viewportHeight="108">
  <path
    android:pathData="M0,0h108v108h-108z">
   <aapt:attr name="android:fillColor">
    <gradient
      android:startY="-1.0078"
      android:startX="54"
      android:endY="106.9922"
      android:endX="54"
      android:type="linear">
     <item android:offset="0" android:color="#FF12C26D"/>
     <item android:offset="1" android:color="#FF0F9453"/>
    </gradient>
   </aapt:attr>
```

```
 </path>
</vector>
```

res/drawable-anydpi-v26/ic_launcher_foreground.xml

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
   android:width="108dp"
   android:height="108dp"
   android:viewportWidth="108"
   android:viewportHeight="108">
  <path
     android:pathData="M38.19,65.92L69.32,65.92a1.2,1.2 0,0 0,1.2 -1.2,0.89 0.89,0 0,0 0,-0.23 17,17
0,0 0,-33.25 0,1.18 1.18,0 0,0 0.9,1.42ZM74.53,69.05a0.85,0.85 0,0 0,-0.78 -0.84L34,68.21a0.85,0.85
0,0 0,-0.77 0.84v2.82a0.84,0.84 0,0 0,0.77 0.86L73.78,72.73a0.85,0.85 0,0 0,0.77 -
0.86L74.55,70.65C74.55,70.24 74.56,69.58 74.53,69.05ZM52.08,49h3.59a1.86,1.86 0,0 0,0 -
3.72L52.08,45.28a1.86,1.86 0,0 0,0 3.72ZM53.87,39.81a1.19,1.19 0,0 0,1.19 -
1.19L55.06,32.87a1.19,1.19 0,0 0,-2.38 0v5.71a1.19,1.19 0,0 0,1.19 1.19h0ZM61.69,41l4.62,-
3.35a1.19,1.19 0,1 0,-1.4 -1.93L60.29,39A1.2,1.2 0,0 0,60 40.67a1.18,1.18 0,0 0,1.66
0.26ZM41.69,37.65L46.31,41a1.2,1.2 0,0 0,1.35 -2L43,35.66a1.19,1.19 0,0 0,-1.66 0.26,1.2 1.2,0 0,0
0.3,1.67Z"
     android:fillColor="#FFFFFF"/>
</vector>
```

Obrazy bitmapowe powinny również zostać automatycznie wygenerowane przez Android Studio w tych lokalizacjach:

```
res/mipmap-mdpi/ic_launcher.png
res/mipmap-mdpi/ic_launcher_round.png
res/mipmap-hdpi/ic_launcher.png
res/mipmap-hdpi/ic_launcher_round.png
res/mipmap-xhdpi/ic_launcher.png
res/mipmap-xhdpi/ic_launcher_round.png
res/mipmap-xxdpi/ic_launcher.png
res/mipmap-xxdpi/ic_launcher_round.png
res/mipmap-xxxdpi/ic_launcher.png
res/mipmap-xxxdpi/ic_launcher_round.png
```

8. Podsumowanie

- Umieść pliki ikon aplikacji w mipmapkatalogach zasobów.
- Udostępniaj różne wersje obrazu bitmapowego ikony aplikacji w każdym zasobniku gęstości (mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi), aby zapewnić zgodność wsteczną ze starszymi wersjami systemu Android.

- Dodaj kwalifikatory zasobów do katalogów zasobów, aby określić zasoby, które powinny być używane na urządzeniach o określonej konfiguracji (np v26.).
- Rysunki wektorowe to implementacja grafiki wektorowej w systemie Android. Są one zdefiniowane w XML jako zbiór punktów, linii i krzywych wraz z powiązanymi informacjami o kolorze. Rysunki wektorowe można skalować pod kątem dowolnej gęstości bez utraty jakości.
- Ikony adaptacyjne zostały wprowadzone na platformę Android w API 26. Składają się z warstwy pierwszego planu i tła, które spełniają określone wymagania, dzięki czemu ikona aplikacji wygląda wysokiej jakości na wielu urządzeniach z różnymi maskami OEM.
- Użyj Image Asset Studio w Android Studio, aby utworzyć starsze i adaptacyjne ikony dla swojej aplikacji.

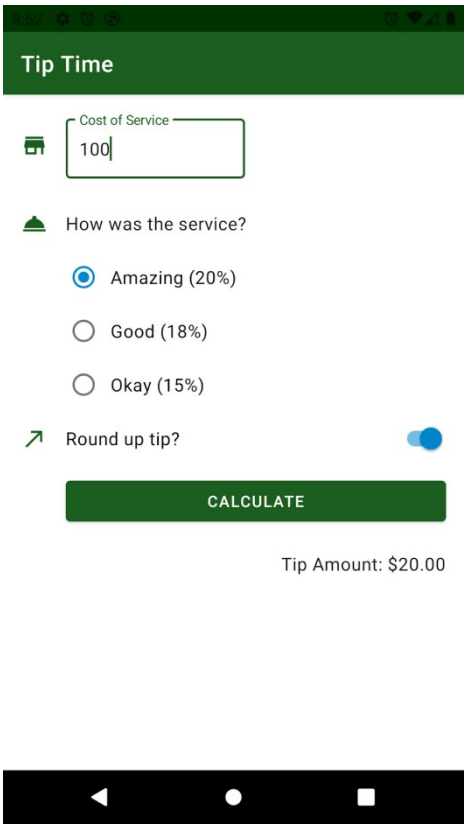
9. Dowiedz się więcej

- Wytyczne projektowe dotyczące ikon Androida
- Ikony adaptacyjne
- Zrozumienie ikon adaptacyjnych Androida
- Projektowanie ikon adaptacyjnych
- Wdrażanie ikon adaptacyjnych
- Adaptacyjna aplikacja do zabaw z ikonami
- Twórz ikony Adaptive i Legacy Launcher
- Obsługa różnych gęstości pikseli
- Umieść ikony aplikacji w katalogach mipmap
- Przegląd rysunków wektorowych
- VectorDrawableklasa

Stwórz bardziej dopracowane wrażenia użytkownika

1. Zanim zaczniesz

Jak już wiesz z wcześniejszych ćwiczeń z programowania, Material to system projektowania stworzony przez Google z wytycznymi, komponentami i narzędziami, które wspierają najlepsze praktyki projektowania interfejsu użytkownika. Podczas tego ćwiczenia z kodowania zaktualizujesz aplikację kalkulatora napiwków (z poprzednich ćwiczeń z kodowania), aby zapewnić bardziej dopracowane wrażenia użytkownika, jak widać na ostatnim zrzucie ekranu poniżej. Przetestujesz również aplikację w kilku dodatkowych scenariuszach, aby zapewnić jak najpłynniejsze wrażenia użytkownika.



Warunki wstępne

- Znajomość popularnych widżetów interfejsu użytkownika, takich jak `TextView`, `ImageView`, `Button`, `EditText`, `RadioButton`, `RadioGroupi``Switch`
- Zapoznanie się z `ConstraintLayout`widokami potomnymi i pozycjonowanie ich przez ustawienie ograniczeń
- Wygodny w modyfikowaniu układów XML
- Świadomy różnicy między obrazami bitmapowymi a rysunkami wektorowymi
- Może ustawić atrybuty motywu w motywie
- Możliwość włączenia ciemnego motywu na urządzeniu
- Wcześniej zmodyfikowano `build.gradle`plik aplikacji pod kątem zależności projektu

Czego się nauczysz

- Jak używać komponentów Material Design w swojej aplikacji

- Jak używać ikon materiałów w aplikacji, importując je z Image Asset Studio
- Jak tworzyć i stosować nowe style
- Jak ustawić inne atrybuty motywu oprócz koloru?

Co zbudujesz

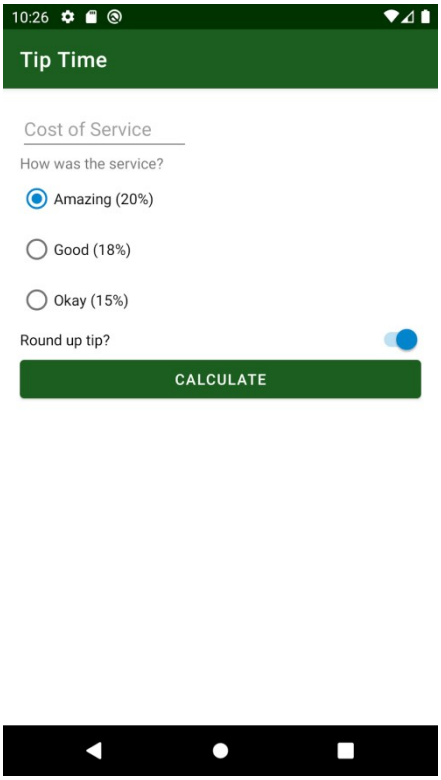
- Dopracowana aplikacja do kalkulatora napiwków, która jest zgodna z zalecanymi najlepszymi praktykami dotyczącymi interfejsu użytkownika

Czego potrzebujesz

- Komputer z zainstalowaną najnowszą stabilną wersją Android Studio
- Kod dla aplikacji **Tip Time** z ukończenia poprzednich ćwiczeń z programowania z tej ścieżki i tej ścieżki

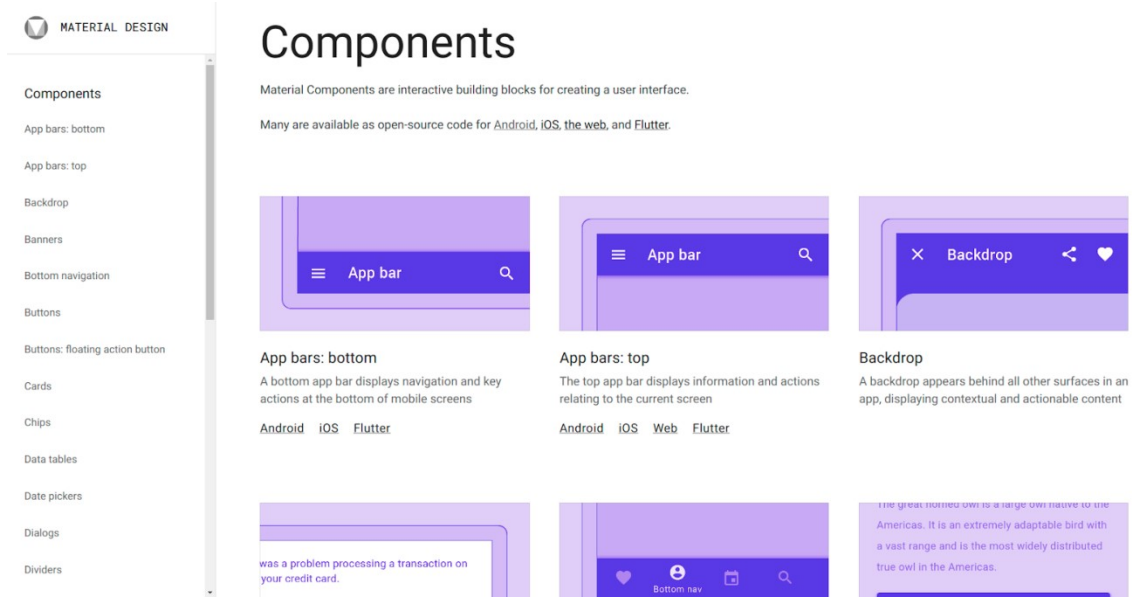
2. Przegląd aplikacji startowej

W ramach wcześniejszych ćwiczeń z programowania zbudowałeś aplikację **Tip Time** , która jest aplikacją kalkulatora napiwków z opcjami dostosowywania napiwków. Interfejs Twojej aplikacji wygląda obecnie tak, jak na poniższym zrzucie ekranu. Funkcjonalność działa, ale wygląda bardziej jak prototyp. Pola nie są do końca wyrównane wizualnie. Zdecydowanie jest miejsce na ulepszenia pod względem bardziej spójnego stylu i odstępów, a także wykorzystania komponentów Material Design.



3. Komponenty materiałowe

Składniki Material to typowe widżety interfejsu użytkownika, które ułatwiają implementację stylów Material w aplikacji. Dokumentacja zawiera informacje dotyczące używania i dostosowywania komponentów Material Design. Istnieją ogólne wytyczne dotyczące projektowania materiałów dla każdego składnika oraz wskazówki dotyczące platformy Android dla składników dostępnych w systemie Android. Diagramy oznaczone etykietą zapewniają wystarczającą ilość informacji, aby odtworzyć komponent, jeśli akurat nie istnieje na wybranej platformie.



Korzystając z Material Components, Twoja aplikacja będzie działać w bardziej spójny sposób wraz z innymi aplikacjami na urządzeniu użytkownika. W ten sposób wzorce interfejsu użytkownika wyuczone w jednej aplikacji można przenieść do następnej. Dzięki temu użytkownicy będą mogli znacznie szybciej nauczyć się korzystania z Twojej aplikacji. Zaleca się używanie komponentów materiałowych , gdy tylko jest to możliwe (w przeciwieństwie do widżetów innych niż materiałowe). Są również bardziej elastyczne i konfigurowalne, o czym dowiesz się w następnym zadaniu.
Bibliotekę Material Design Components (MDC) należy uwzględnić jako zależność w projekcie. Ta linia powinna już być domyślnie obecna w Twoim projekcie. W pliku aplikacji build.gradleupewnij się, że ta zależność jest uwzględniona w najnowszej wersji biblioteki. Aby uzyskać więcej informacji, zobacz stronę Rozpocznij w witrynie Materiał.

app/build.gradle

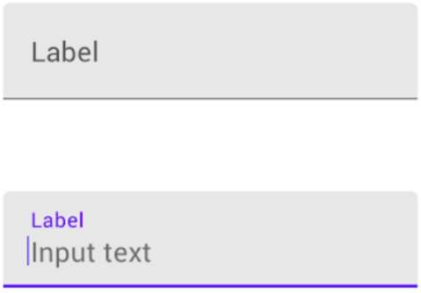
```
dependencies {
  ...
  implementation 'com.google.android.material:material:<version>'
}
```

Pola tekstowe

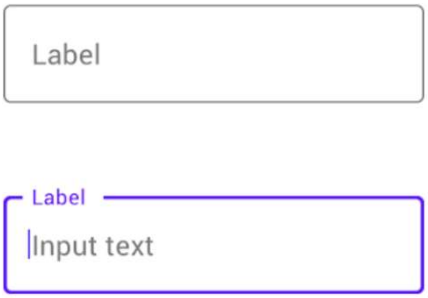
W aplikacji kalkulatora napiwków u góry układu znajduje się obecnie EditTextpole na koszt usługi. To EditTextpole działa, ale nie jest zgodne z najnowszymi wytycznymi Material Design dotyczącymi wyglądu i zachowania pól tekstowych.

W przypadku każdego nowego komponentu, którego chcesz użyć, zacznij od zapoznania się z nim w witrynie Materiał. W przewodniku dotyczącym pól tekstowych istnieją dwa rodzaje pól tekstowych:

Wypełnione pole tekstowe



Obrysowane pole tekstowe



Aby utworzyć pole tekstowe, jak pokazano powyżej, użyj `TextInputLayout`znaku z dołączonym `TextInputEditText`z biblioteki MDC. Pole tekstowe Materiał można łatwo dostosować do:

- Wyświetlaj tekst wejściowy lub etykietę, która jest zawsze widoczna
- Wyświetl ikonę w polu tekstowym
- Wyświetl pomocnika lub komunikaty o błędach

W pierwszym zadaniu tego ćwiczenia z programowania zamienisz koszt usługi `EditText`na pole tekstowe Materiał (składające się z a `TextInputLayout`i `TextInputEditText`).

1. Otwórz aplikację **Tip Time** w Android Studio i przejdź do `activity_main.xml`pliku układu. Powinien zawierać a `ConstraintLayout`wraz z układem kalkulatora napiwków.
2. Aby zobaczyć przykład tego, jak wygląda kod XML dla pola tekstowego Materiał, wróć do wskazówek dotyczących systemu Android dla pól tekstowych . Powinieneś zobaczyć fragmenty takie jak ten:

```
<com.google.android.material.textfield.TextInputLayout
  android:id="@+id/textField"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:hint="@string/label">

  <com.google.android.material.textfield.TextInputEditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
  />
```

```
</com.google.android.material.textfield.TextInputLayout>
```

3. Po obejrzeniu tego przykładu wstaw pole tekstowe Materiał jako pierwsze dziecko `ConstraintLayout`(przed `EditText`polem). `EditText`W późniejszym etapie pozbędziesz się pola.

   Możesz wpisać to w Android Studio i użyć autouzupełniania, aby było łatwiej. Lub możesz skopiować przykładowy kod XML ze strony dokumentacji i wkleić go do swojego układu w ten sposób. Zwróć uwagę, jak `TextInputLayout`ma widok potomny, `TextInputEditText`. Pamiętaj, że wielokropek (...) służy do skracania fragmentów kodu, dzięki czemu możesz skupić się na faktycznie zmienionych wierszach XML.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
  ...>

  <com.google.android.material.textfield.TextInputLayout
    android:id="@+id/textField"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/label">

    <com.google.android.material.textfield.TextInputEditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
    />

  </com.google.android.material.textfield.TextInputLayout>

  <EditText
    android:id="@+id/cost_of_service" ... />

  ...
```

   Oczekuje się, że zobaczysz błędy w `TextInputLayout`elemencie. Ten widok nie został jeszcze odpowiednio ograniczony w widoku nadrzędnym `ConstraintLayout`. Również zasób tekstowy nie jest rozpoznawany. W kolejnych krokach naprawisz te błędy.

```
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/textField"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/label">
```

4. Dodaj ograniczenia pionowe i poziome do pola tekstowego, aby prawidłowo umieścić je w polu nadrzędnym `ConstraintLayout`. Ponieważ nie zostały jeszcze usunięte `EditText`, wytnij i wklej następujące atrybuty z `EditText`i umieść je w `TextInputLayout`: ograniczenia, identyfikator zasobu `cost_of_service`, szerokość `160dp`układu , wysokość układu `wrap_content`i tekst podpowiedzi `@string/cost_of_service`.

   ...

```
<com.google.android.material.textfield.TextInputLayout
  android:id="@+id/cost_of_service"
  android:layout_width="160dp"
  android:layout_height="wrap_content"
  android:hint="@string/cost_of_service"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toTopOf="parent">

  <com.google.android.material.textfield.TextInputEditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>

</com.google.android.material.textfield.TextInputLayout>

...
```

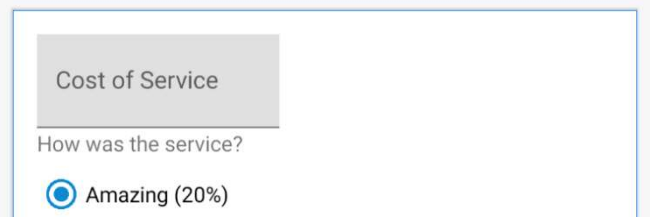
Możesz zobaczyć błąd, że `cost_of_service`identyfikator jest taki sam jak identyfikator zasobu `EditText`, ale możesz na razie zignorować ten błąd. (`EditText`zostanie usunięty w kilku krokach).

5. Następnie upewnij się, że `TextInputEditText`element ma wszystkie odpowiednie atrybuty. Wytnij i wklej typ danych wejściowych z `EditText`na `TextInputEditText`. Zmień `TextInputEditText`identyfikator zasobu elementu na `cost_of_service_edit_text`.

```
<com.google.android.material.textfield.TextInputLayout ... >

  <com.google.android.material.textfield.TextInputEditText
    android:id="@+id/cost_of_service_edit_text"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="numberDecimal" />

</com.google.android.material.textfield.TextInputLayout>
```

Szerokość `match_parent`i wysokość `wrap_content`jest w porządku bez zmian. Podczas ustawiania szerokości `match_parent`, `TextInputEditText`będzie miał taką samą szerokość jak jego rodzic `TextInputLayout`, którym jest `160dp`.

6. Po skopiowaniu wszystkich istotnych informacji z `EditText`, przejdź dalej i usuń `EditText`z układu.
7. W widoku **Projekt** układu powinieneś zobaczyć ten podgląd. Pole Koszt usługi wygląda teraz jak pole tekstowe Materiał.

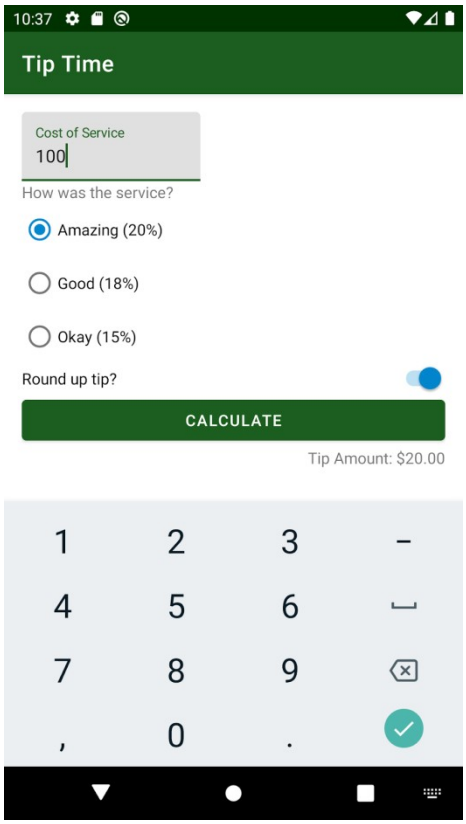


8. Nie możesz jeszcze uruchomić aplikacji, ponieważ w `MainActivity.kt`pliku w `calculateTip()`metodzie jest błąd. Przypomnij sobie z wcześniejszego ćwiczenia programowania, które przy włączonym powiązaniu widoku dla projektu, system Android tworzy właściwości w obiekcie powiązania na podstawie nazwy identyfikatora zasobu. Pole, z którego pobraliśmy koszt usługi, zmieniło się w układzie XML, więc kod Kotlina należy odpowiednio zaktualizować.

   Teraz będziesz pobierać dane wejściowe użytkownika z `TextInputEditText`elementu o identyfikatorze zasobu `cost_of_service_edit_text`. W `MainActivity`, użyj , `binding.costOfServiceEditText`aby uzyskać dostęp do przechowywanego w nim ciągu tekstowego. Reszta `calculateTip()`metody może pozostać taka sama.

```
private fun calculateTip() {
    // Get the decimal value from the cost of service text field
    val stringInTextField = binding.costOfServiceEditText.text.toString()
    val cost = stringInTextField.toDoubleOrNull()

    ...
}
```

9. Świetna robota! Teraz uruchom aplikację i sprawdź, czy nadal działa. Zwróć uwagę, że etykieta „Koszt usługi” pojawia się teraz nad danymi wejściowymi podczas pisania. Wskazówka powinna nadal obliczyć zgodnie z oczekiwaniami.



Przełączniki

W wytycznych Material Design znajdują się również wskazówki dotyczące przełączników . Przełącznik to widżet, w którym można włączać i wyłączać ustawienie.

1. Zapoznaj się ze wskazówkami Androida dotyczącymi przełączników materiałów . Poznasz `SwitchMaterial`widżet (z biblioteki MDC), który zapewnia stylizację materiału dla przełączników. Jeśli będziesz przewijać przewodnik, zobaczysz kilka przykładów XML.

2. Aby użyć SwitchMaterial, musisz jawnie określić SwitchMaterialw układzie i użyć w pełni kwalifikowanej nazwy ścieżki.

   W activity_main.xmlukładzie zmień tag XML
   z Switchna com.google.android.material.switchmaterial.SwitchMaterial.

   ...

   ```
   <com.google.android.material.switchmaterial.SwitchMaterial
     android:id="@+id/round_up_switch"
     android:layout_width="0dp"
     android:layout_height="wrap_content" ... />
   ```

   ...

3. Uruchom aplikację, aby sprawdzić, czy nadal się kompiluje. Zdarza się, że w aplikacji nie ma widocznych zmian. Jednak zaletą korzystania SwitchMaterialz biblioteki MDC (zamiast Switchz platformy Android) jest to, że gdy implementacja biblioteki dla SwitchMaterialzostanie zaktualizowana (np. zmienią się wytyczne Material Design), otrzymasz zaktualizowany widżet za darmo bez żadnych wymaganych zmian z Twojej strony. Pomaga to zabezpieczyć Twoją aplikację w przyszłości.

   W tym momencie widziałeś dwa przykłady tego, jak Twój interfejs użytkownika może skorzystać z używania gotowych komponentów Material Design i jak to przybliża Twoją aplikację do wytycznych dotyczących materiałów. Pamiętaj, że zawsze możesz zapoznać się z innymi komponentami Material Design dostępnymi w systemie Android na tej stronie .

4. Ikony

Ikony to symbole, które mogą pomóc użytkownikom zrozumieć interfejs użytkownika, komunikując wizualnie zamierzoną funkcję. Często czerpią inspirację z obiektów w świecie fizycznym, których oczekuje się od użytkownika. Projektowanie ikon często redukuje poziom szczegółowości do minimum wymaganego, aby były znane użytkownikowi. Na przykład ołówek w świecie fizycznym służy do pisania, więc jego odpowiednik w postaci ikony zwykle wskazuje na tworzenie, dodawanie lub edycję elementu.

Zdjęcie autorstwa Angeliny Litvin na Unsplash

Czasami ikony są połączone z przestarzałymi obiektami świata fizycznego, tak jak w przypadku ikony dyskietki. Ta ikona jest wszechobecnym wskaźnikiem zapisywania pliku lub rekordu bazy danych; jednakże, chociaż dyskietki zostały spopularyzowane w latach 70., przestały być powszechne po 2000 roku. Jednak ich ciągłe używanie dzisiaj świadczy o tym, jak silny obraz może przekroczyć żywotność swojej fizycznej formy.

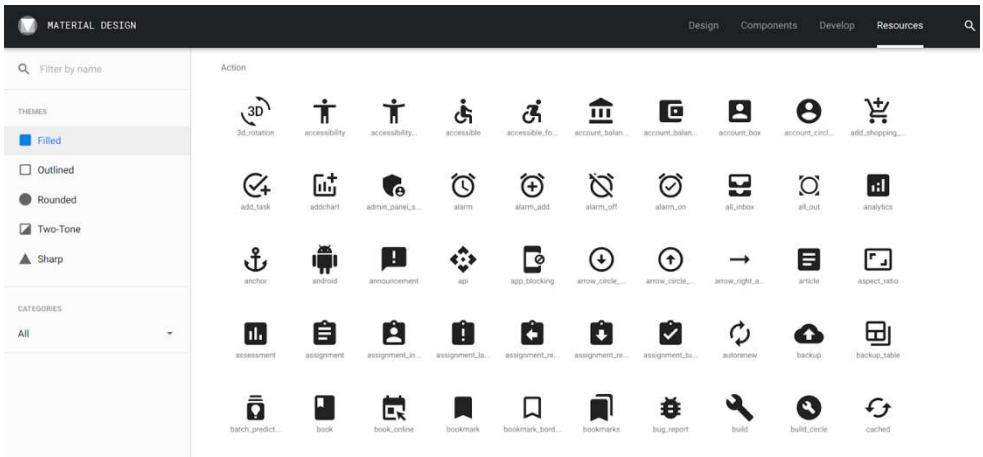
Zdjęcie Vincenta Botta na Unsplash

Reprezentowanie ikon w Twojej aplikacji

W przypadku ikon w aplikacji zamiast udostępniać różne wersje obrazu bitmapowego dla różnych gęstości ekranu, zalecaną praktyką jest użycie obiektów do rysowania wektorów. Rysunki wektorowe są reprezentowane jako pliki XML, które przechowują instrukcje dotyczące tworzenia obrazu, zamiast zapisywania rzeczywistych pikseli, które składają się na ten obraz. Rysunki wektorowe można skalować w górę lub w dół bez utraty jakości wizualnej lub zwiększania rozmiaru pliku.

Dostarczone ikony

Material Design zapewnia szereg ikon ułożonych w wspólne kategorie dla większości Twoich potrzeb. Zobacz listę ikon .



Ikony te można również narysować za pomocą jednego z pięciu motywów (wypełnione, obrysowane, zaokrąglone, dwutonowe i ostre) i mogą być zabarwione kolorami.

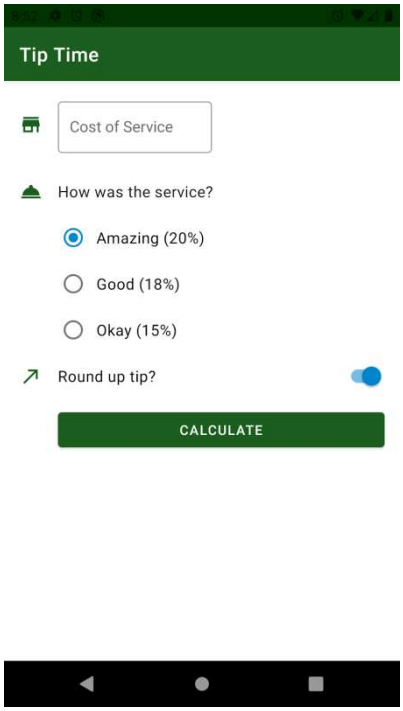
Wypełniony	Przedstawione	Bułczasty	Dwutonowy	Ostry

Dodawanie ikon

W tym zadaniu dodasz do aplikacji trzy wektorowe ikony do rysowania:

1. Ikona obok pola tekstowego kosztu usługi
2. Ikona obok pytania serwisowego
3. Ikona obok monitu o zaokrąglenie w górę

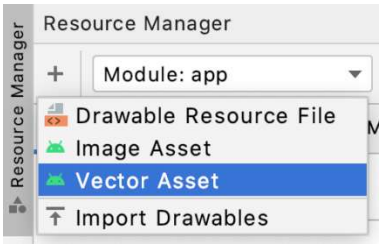
Poniżej znajduje się zrzut ekranu ostatecznej wersji aplikacji. Po dodaniu ikon dostosujesz układ, aby dostosować rozmieszczenie tych ikon. Zwróć uwagę, jak pola i przycisk obliczania są przesunięte w prawo, z dodatkiem ikon.



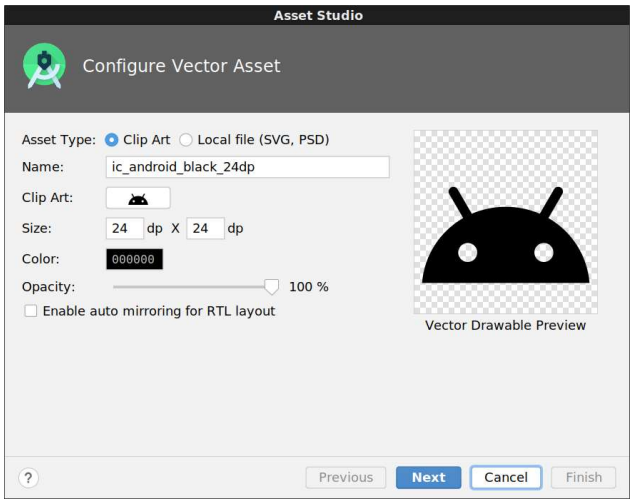
Dodaj zasoby wektorowe do rysowania

Możesz tworzyć te ikony jako wektory do rysowania bezpośrednio z **Asset Studio** w Android Studio.

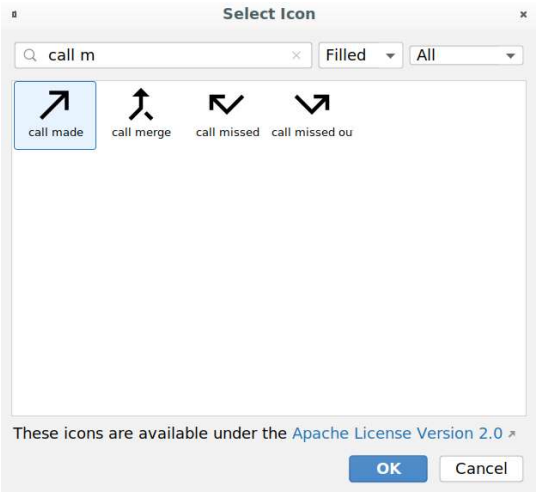
1. Otwórz zakładkę **Menedżer zasobów** znajdującą się po lewej stronie okna aplikacji.
2. Kliknij ikonę + i wybierz **Zasób wektorowy.**



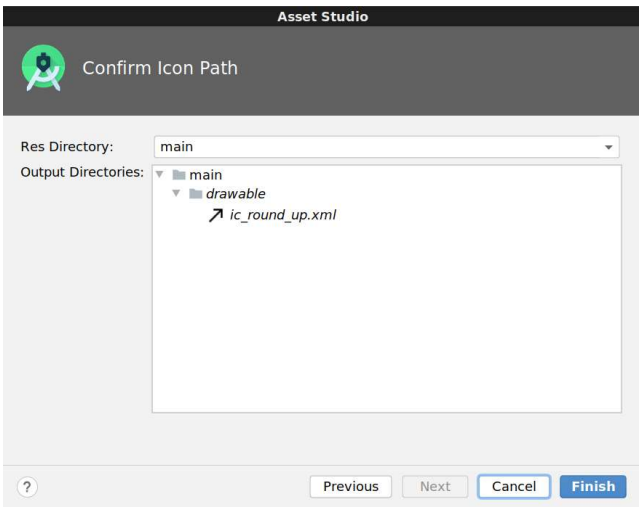
3. Upewnij się, że w polu **Typ zasobu** jest zaznaczony przycisk opcji **Clip Art .**



4. Kliknij przycisk obok **Clip Art:,** aby wybrać inny obraz clipart. W wyświetlonym monicie wpisz „połączenie wykonane” w wyświetlonym oknie. Będziesz używać tej ikony strzałki jako opcji zaokrąglania napiwku. Wybierz go i kliknij **OK** .



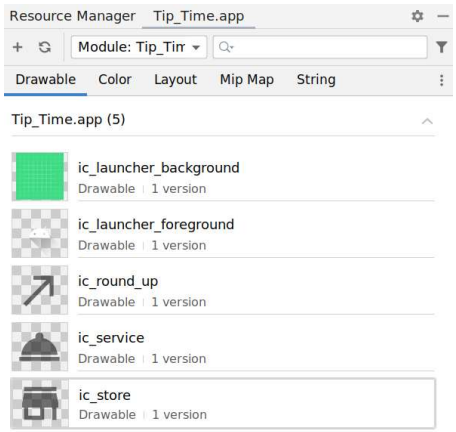
5. Zmień nazwę ikony na ic_round_up. (Zaleca się używanie przedrostka ic_ podczas nazywania plików ikon.) Możesz pozostawić Rozmiar jako 24 dp x 24 dp i Kolor jako czarny 000000.
6. Kliknij **Dalej** .
7. Zaakceptuj domyślną lokalizację katalogu i kliknij przycisk **Zakończ** .



8. Powtórz kroki 2–7 dla pozostałych dwóch ikon:

- **Ikona pytania o usługę: wyszukaj ikonę** „obsługa pokoju”, zapisz ją jako `ic_service`.
- **Ikona kosztu usługi: wyszukaj ikonę** „sklep”, zapisz ją jako `ic_store`.

9. Gdy skończysz, **Menedżer zasobów** będzie wyglądał jak na poniższym zrzucie ekranu. Będziesz także mieć te trzy wektory do rysowania (`ic_round_up`, `ic_service`i `ic_store`) wymienione w twoim `res/drawable`folderze.



Obsługa starszych wersji Androida

Właśnie dodano do aplikacji elementy wektorowe do rysowania, ale należy pamiętać, że obsługa elementów wektorowych do rysowania na platformie Android nie została dodana do systemu Android 5,0 (poziom interfejsu API 21) .

W zależności od konfiguracji projektu minimalna wersja pakietu SDK dla aplikacji Tip Time to API 19. Oznacza to, że aplikacja może działać na urządzeniach z systemem Android w wersji 19 lub nowszej.

Aby Twoja aplikacja działała w tych starszych wersjach Androida (tzw. kompatybilność wsteczna), dodaj `vectorDrawables`element do `build.gradle`pliku aplikacji. Umożliwia to korzystanie z obiektów do rysowania wektorów w wersjach platformy mniejszych niż API 21, w przeciwieństwie do konwersji do plików PNG podczas kompilowania projektu. Zobacz więcej szczegółów tutaj .

`app/build.gradle`

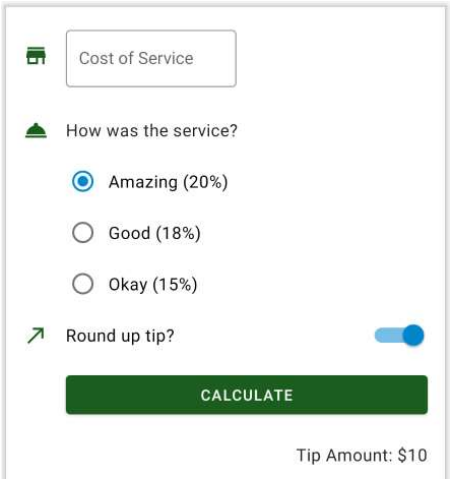
```
android {
 defaultConfig {
```

```
    ...
    vectorDrawables.useSupportLibrary = true
  }
  ...
}
```

Po prawidłowym skonfigurowaniu projektu możesz teraz przejść do dodawania ikon do układu.

Wstaw ikony i pozycjonuj elementy

Będziesz używać `ImageViews`do wyświetlania ikon w aplikacji. Tak będzie wyglądał Twój ostateczny interfejs użytkownika.



1. Otwórz `activity_main.xml`układ.
2. Najpierw umieść ikonę sklepu obok pola tekstowego kosztu usługi. Wstaw nowy `ImageView`jako pierwsze dziecko `ConstraintLayout`, przed `TextInputLayout`.

```
<androidx.constraintlayout.widget.ConstraintLayout
  ...>

  <ImageView
    android:layout_width=""
    android:layout_height=""

  <com.google.android.material.textfield.TextInputLayout
    android:id="@+id/cost_of_service"
    ...
```

3. Ustaw odpowiednie atrybuty, `ImageView`aby przytrzymać `ic_store`ikonę. Ustaw identyfikator na `icon_cost_of_service`. Ustaw `app:srcCompat`atrybut na zasób do rysowania `@drawable/ic_store`, a zobaczysz podgląd ikony obok tego wiersza kodu XML.

   Również ustawiony, `android:importantForAccessibility="no"`ponieważ ten obraz służy wyłącznie do celów dekoracyjnych.

```
<ImageView
  android:id="@+id/icon_cost_of_service"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:importantForAccessibility="no"
  app:srcCompat="@drawable/ic_store" />
```

Oczekuje się, że pojawi się błąd, `ImageView`ponieważ widok nie jest jeszcze ograniczony. Naprawisz to później.

4. Ustaw `icon_cost_of_service`w dwóch krokach. Najpierw dodaj ograniczenia na `ImageView`(ten krok), a następnie zaktualizuj ograniczenia na `TextInputLayout`sąsiednim (krok 5). Ten diagram pokazuje, jak należy skonfigurować ograniczenia.

W przypadku `ImageView`chcesz, aby jego początkowa krawędź była związana z początkową krawędzią widoku rodzica (`app:layout_constraintStart_toStartOf="parent"`).

Ikona pojawia się wyśrodkowana w pionie w porównaniu z polem tekstowym obok niej, więc ogranicz górną część tej ikony `ImageView`(`layout_constraintTop_toTopOf`) do górnej części pola tekstowego. Ogranicz dolną część tego `ImageView`(`layout_constraintBottom_toBottomOf`) do dolnej części pola tekstowego. Aby odwołać się do pola tekstowego, użyj identyfikatora zasobu `@id/cost_of_service`. Domyślnym zachowaniem jest to, że gdy dwa więzy są stosowane do widżetu w tym samym wymiarze (takim jak więz górny i dolny), więzy są stosowane jednakowo. Powoduje to, że ikona zostaje wyśrodkowana w pionie w stosunku do kosztu pola serwisowego.

```
<ImageView
  android:id="@+id/icon_cost_of_service"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:importantForAccessibility="no"
  app:srcCompat="@drawable/ic_store"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toTopOf="@id/cost_of_service"
  app:layout_constraintBottom_toBottomOf="@id/cost_of_service" />
```

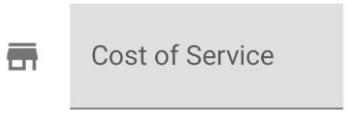
Ikona i pole tekstowe nadal nakładają się na siebie w widoku **Projekt** . Zostanie to naprawione w następnym kroku.

5. Przed dodaniem ikony pole tekstowe znajdowało się na początku rodzica. Teraz trzeba go przesunąć w prawo. Zaktualizuj ograniczenia w `cost_of_service`polu tekstowym w odniesieniu do `icon_cost_of_service`.

Początkowa krawędź TextInputLayoutpowinna być powiązana z końcową krawędzią ImageView(@id/icon_cost_of_service). Aby dodać trochę odstępów między dwoma widokami, dodaj margines początkowy 16dpna TextInputLayout.

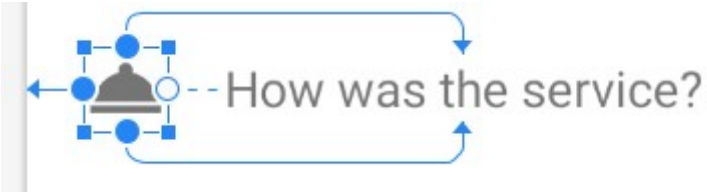
```
<com.google.android.material.textfield.TextInputLayout
  android:id="@+id/cost_of_service"
  ...
  android:layout_marginStart="16dp"
  app:layout_constraintStart_toEndOf="@id/icon_cost_of_service">

  <com.google.android.material.textfield.TextInputEditText ... />

</com.google.android.material.textfield.TextInputLayout>
```

Po tych wszystkich zmianach ikona powinna być prawidłowo umieszczona obok pola tekstowego.



6. Następnie wstaw ikonę dzwonka usługi obok „Jaka była usługa?” TextView. Chociaż możesz zadeklarować w ImageViewdowolnym miejscu w ConstraintLayout, układ XML będzie łatwiejszy do odczytania, jeśli wstawisz nowy ImageVieww układzie XML po TextInputLayout, ale przed service_question TextView.

   Dla nowego ImageView, przypisz mu identyfikator zasobu @+id/icon_service_question. Ustaw odpowiednie ograniczenia na ImageViewpytanie serwisowe i TextView.

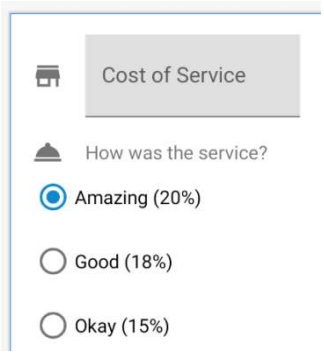


   Dodaj także 16dpgórny margines, aby service_question TextViewbyło więcej miejsca w pionie między pytaniem o usługę a polem tekstowym kosztu usługi nad nim.

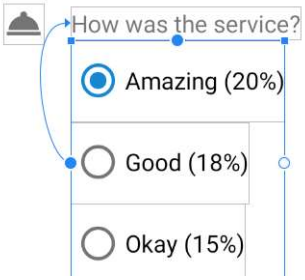
```
...

 <ImageView
    android:id="@+id/icon_service_question"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:importantForAccessibility="no"
    app:srcCompat="@drawable/ic_service"
```

```
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="@id/service_question"
    app:layout_constraintBottom_toBottomOf="@id/service_question" />

  <TextView
    android:id="@+id/service_question"
    ...
    android:layout_marginTop="16dp"
    app:layout_constraintStart_toStartOf="@id/cost_of_service"
    app:layout_constraintTop_toBottomOf="@id/cost_of_service"/>

...
```

7. W tym momencie widok **Projekt** powinien wyglądać tak. Koszt pola serwisowego i pytania serwisowego (oraz odpowiadające im ikony) wyglądają świetnie, ale przyciski opcji wyglądają teraz nie na miejscu. Nie są wyrównane w pionie z treścią powyżej.



8. Popraw rozmieszczenie przycisków opcji, przesuwając je w prawo pod pytaniem serwisowym. Oznacza to aktualizację `RadioGroup`ograniczenia. Powiąż początkową krawędź z `RadioGroup`początkową krawędzią `service_question` `TextView`. Wszystkie inne atrybuty na `RadioGroup`puszce pozostaną takie same.

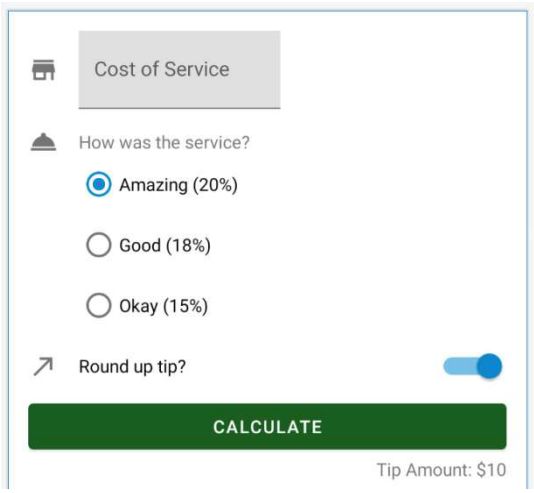



```
...

<RadioGroup
  android:id="@+id/tip_options"
  ...
  app:layout_constraintStart_toStartOf="@id/service_question">

...
```

9. Następnie kontynuuj dodawanie ic_round_upikony do układu obok „Wskazówki podsumowującej?” przełącznik. Spróbuj zrobić to na własną rękę, a jeśli utkniesz, możesz zapoznać się z poniższym kodem XML. Możesz przypisać nowy ImageViewidentyfikator zasobu icon_round_up.
10. W formacie XML układu wstaw nowy ImageViewpo widżecie, RadioGroupale przed SwitchMaterialwidżetem.
11. Przypisz ImageViewidentyfikator zasobu icon_round_upi ustaw srcCompatikonę do rysowania @drawable/ic_round_up. Ogranicz początek ImageViewelementu nadrzędnego do początku elementu nadrzędnego, a także wyśrodkuj ikonę w pionie względem elementu SwitchMaterial.
12. Zaktualizuj SwitchMaterial, aby znajdował się obok ikony i miał 16dpmargines początkowy. Tak powinien wyglądać wynikowy XML icon_round_upi round_up_switch.

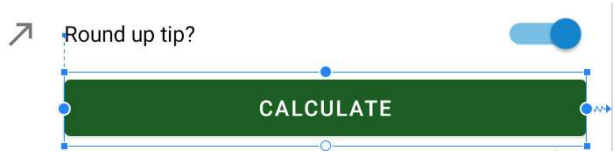
```
...

 <ImageView
    android:id="@+id/icon_round_up"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:importantForAccessibility="no"
    app:srcCompat="@drawable/ic_round_up"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="@id/round_up_switch"
    app:layout_constraintBottom_toBottomOf="@id/round_up_switch" />

  <com.google.android.material.switchmaterial.SwitchMaterial
    android:id="@+id/round_up_switch"
    ...
    android:layout_marginStart="16dp"
    app:layout_constraintStart_toEndOf="@id/icon_round_up" />

...
```

13. Widok **Projekt** powinien wyglądać tak. Wszystkie trzy ikony są prawidłowo ustawione.



14. Jeśli porównasz to z końcowym zrzutem ekranu aplikacji, zauważysz, że przycisk obliczania jest również przesunięty, aby wyrównać w pionie z kosztem pola usługi, pytaniem serwisowym, opcjami przycisków

opcji i pytaniem o zaokrąglenie w górę. Osiągnij ten wygląd, ograniczając początek przycisku obliczania do początku `round_up_switch`. Dodaj także `8dp`pionowy margines między przyciskiem Oblicz a przełącznikiem nad nim.

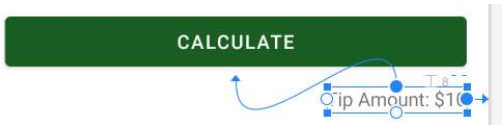


...

```
<Button
  android:id="@+id/calculate_button"
  ...
  android:layout_marginTop="8dp"
  app:layout_constraintStart_toStartOf="@id/round_up_switch" />
```

...

15. Ostatnia, ale nie mniej ważna pozycja `tip_result`, dodając `8dp`górny margines do `TextView`.

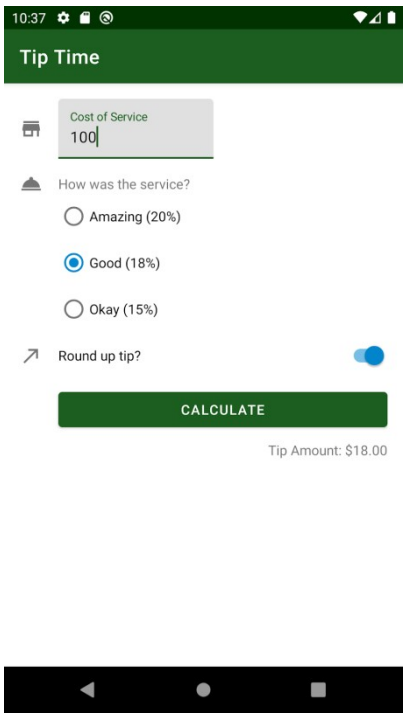


...

```
<TextView
  android:id="@+id/tip_result"
  ...
  android:layout_marginTop="8dp" />
```

...

16. To było wiele kroków! Świetna robota przy ich przejściu krok po kroku. Prawidłowe wyrównanie elementów w układzie wymaga wiele uwagi do szczegółów, ale sprawia, że efekt końcowy wygląda znacznie lepiej! Uruchom aplikację i powinna wyglądać jak na poniższym zrzucie ekranu. Dzięki wyrównaniu w pionie i zwiększeniu odstępów między elementami nie są one tak stłoczone.



Jeszcze nie skończyłeś! Być może zauważyłeś, że rozmiar i kolor czcionki pytania serwisowego oraz kwota napiwku wyglądają inaczej niż tekst w przyciskach radiowych i przełączniku. Sprawmy, aby były one spójne w następnym zadaniu, używając stylów i motywów.

5. Style i motywy

Styl to zbiór wartości atrybutów widoku dla jednego typu widżetu . Na przykład TextViewstyl może określać kolor czcionki, rozmiar czcionki i kolor tła, aby wymienić tylko kilka. Wyodrębniając te atrybuty do stylu, możesz łatwo zastosować styl do wielu widoków w układzie i zachować go w jednym miejscu.

W tym zadaniu najpierw utworzysz style dla widoku tekstu, przycisku radiowego i widżetów przełączania.

Twórz style

1. Utwórz nowy plik o nazwie styles.xmlw katalogu **res > values** , jeśli jeszcze taki nie istnieje. Utwórz go, klikając prawym przyciskiem myszy katalog **wartości** i wybierając **Nowy > Plik zasobów wartości** . Nazwij to styles.xml. Nowy plik będzie miał następującą zawartość.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
</resources>
```

2. Utwórz nowy TextViewstyl, aby tekst był spójny w całej aplikacji. Zdefiniuj styl raz w styles.xml, a następnie możesz zastosować go do wszystkich TextViewsw układzie. Chociaż możesz zdefiniować styl od podstaw, możesz rozszerzyć istniejący TextViewstyl z biblioteki MDC.

   Stylizując komponent, powinieneś generalnie wywodzić się ze stylu nadrzędnego używanego typu widżetu. Jest to ważne z dwóch powodów. Po pierwsze, upewnia się, że wszystkie ważne wartości

domyślne są ustawione w Twoim komponencie, a po drugie, Twój styl będzie nadal dziedziczył wszelkie przyszłe zmiany tego stylu nadrzędnego.

Możesz nazwać swój styl, jak chcesz, ale istnieje zalecana konwencja. Jeśli dziedziczysz po nadrzędnym stylu Material, nazwij swój styl równolegle, zastępując `MaterialComponents`go nazwą swojej aplikacji (`TipTime`). Przenosi to zmiany do własnej przestrzeni nazw, co eliminuje możliwość przyszłych konfliktów, gdy Material Components wprowadza nowe style. Przykład:

Nazwa Twojego stylu: `Widget.TipTime.TextView`dziedziczy po stylu nadrzędnym: `Widget.MaterialComponents.TextView`.

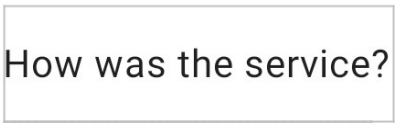
Dodaj to do swojego `styles.xml`pliku pomiędzy `resources`tagami otwierającymi i zamykającymi.

```
<style name="Widget.TipTime.TextView" parent="Widget.MaterialComponents.TextView">
</style>
```

3. Skonfiguruj swój `TextView`styl tak, aby zastępował następujące atrybuty: `android:minHeight`,`android:gravity`,i`android:textAppearance`.

   `android:minHeight`ustawia minimalną wysokość 48dp na `TextView`. Najmniejsza wysokość każdego rzędu powinna wynosić 48 dp zgodnie z wytycznymi Material Design .

   Możesz wyśrodkować tekst w `TextView`pionie, ustawiając `android:gravity`atrybut. (Zobacz zrzut ekranu poniżej.) Grawitacja kontroluje sposób, w jaki zawartość w widoku będzie się pozycjonować. Ponieważ rzeczywista zawartość tekstu nie zajmuje pełnej wysokości 48 dp, wartość `center_vertical`wyśrodkowuje tekst w `TextView`pionie (ale nie zmienia jego położenia w poziomie). Inne możliwe wartości wagi to `center`, `center_horizontal`, `top`i `bottom`. Zachęcamy do wypróbowania innych wartości grawitacji, aby zobaczyć wpływ na tekst.



   Ustaw wartość atrybutu wyglądu tekstu na `?attr/textAppearanceBody1`. TextAppearance to zestaw gotowych stylów dotyczących rozmiaru tekstu, czcionek i innych właściwości tekstu. Aby zapoznać się z innymi możliwymi wyglądami tekstu udostępnianymi przez Materiał, zobacz tę listę skal typów .

```
<style name="Widget.TipTime.TextView" parent="Widget.MaterialComponents.TextView">
  <item name="android:minHeight">48dp</item>
  <item name="android:gravity">center_vertical</item>
  <item name="android:textAppearance">?attr/textAppearanceBody1</item>
</style>
```

4. Zastosuj `Widget.TipTime.TextView`styl do elementu `service_question` `TextView`, dodając atrybut stylu do każdego elementu `TextView`w `activity_main.xml`.

```
<TextView
  android:id="@+id/service_question"
  style="@style/Widget.TipTime.TextView"
  ... />
```

   Przed stylem `TextView`wyglądało to tak z małym rozmiarem czcionki i szarym kolorem czcionki:

Po dodaniu stylu `TextView`wygląda to tak. Teraz `TextView`wygląda to bardziej spójnie z resztą układu.

5. Zastosuj ten sam `Widget.TipTime.TextView`styl do `tip_result` `TextView`.

```
<TextView
  android:id="@+id/tip_result"
  style="@style/Widget.TipTime.TextView"
  ... />
```

Uwaga: Jeśli określisz atrybut w stylu (np. set `android:textSize`to be `18sp`), a także określisz ten sam atrybut w pliku układu (np. set `android:textSize`to be `14sp`), wtedy wartość ustawiona w układzie (`14sp`) zostanie faktycznie zastosowana do tego, co widzisz na ekranie.

6. Ten sam styl tekstu należy zastosować do etykiety tekstowej w przełączniku. Nie możesz jednak ustawić `TextView`stylu w `SwitchMaterial`widżecie. `TextView`style można stosować tylko na `TextViews`. Dlatego stwórz nowy styl dla przełącznika. Atrybuty są takie same w zakresie `minHeight`, `gravity`i `textAppearance`. Różnią się tutaj nazwa stylu i styl nadrzędny, ponieważ dziedziczysz teraz `Switch`styl z biblioteki MDC. Twoja nazwa stylu powinna również odzwierciedlać nazwę stylu nadrzędnego.

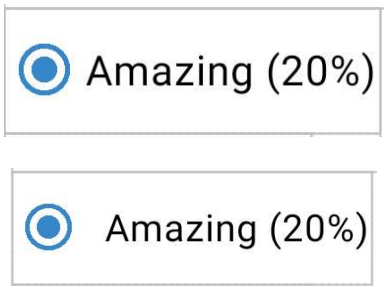
   Nazwa Twojego stylu: `Widget.TipTime.CompoundButton.Switch`dziedziczy po stylu nadrzędnym:`Widget.MaterialComponents.CompoundButton.Switch`.

```
<style name="Widget.TipTime.CompoundButton.Switch"
parent="Widget.MaterialComponents.CompoundButton.Switch">
  <item name="android:minHeight">48dp</item>
  <item name="android:gravity">center_vertical</item>
  <item name="android:textAppearance">?attr/textAppearanceBody1</item>
</style>
```

   Możesz również określić dodatkowe atrybuty specyficzne dla przełączników w tym stylu, ale w Twojej aplikacji nie ma takiej potrzeby.

7. Tekst przycisku opcji to ostatnie miejsce, w którym chcesz się upewnić, że tekst jest spójny wizualnie. Nie można zastosować `TextView`stylu ani `Switch`stylu do `RadioButton`widżetu. Zamiast tego musisz utworzyć nowy styl dla przycisków radiowych. Możesz rozszerzyć `RadioButton`styl biblioteki MDC.

Podczas tworzenia tego stylu dodaj także dopełnienie między tekstem przycisku opcji a grafiką okręgu. paddingStartto nowy atrybut, którego jeszcze nie używałeś. Padding to ilość miejsca między zawartością widoku a granicami widoku. Atrybut ustawia dopełnienie tylko na paddingStartpoczątku komponentu. Zobacz różnicę między 0dp a 8dp paddingStartna przycisku radiowym.




```
<style name="Widget.TipTime.CompoundButton.RadioButton"
parent="Widget.MaterialComponents.CompoundButton.RadioButton">
  <item name="android:paddingStart">8dp</item>
  <item name="android:textAppearance">?attr/textAppearanceBody1</item>
</style>
```

8. (Opcjonalnie) Utwórz dimens.xmlplik, aby poprawić łatwość zarządzania często używanymi wartościami. Możesz utworzyć plik w taki sam sposób, jak w przypadku styles.xmlpowyższego pliku. Wybierz katalog wartości, kliknij prawym przyciskiem myszy i wybierz **Nowy > Plik zasobów wartości** .

   W tej małej aplikacji dwukrotnie powtórzyłeś ustawienie minimalnej wysokości. Na razie jest to z pewnością do opanowania, ale szybko wymknęłoby się spod kontroli, gdybyśmy mieli 4, 6, 10 lub więcej komponentów o tej samej wartości. Pamiętanie o zmianie wszystkich z osobna jest żmudne i podatne na błędy. Możesz utworzyć inny pomocny plik zasobów w **res >** o nazwie wartości dimens.xml, który zawiera wspólne wymiary, które możesz nazwać. Standaryzując wspólne wartości jako nazwane wymiary, ułatwiamy zarządzanie naszą aplikacją. TipTime jest mały, więc nie będziemy go używać poza tym opcjonalnym krokiem. Jednak przy bardziej złożonych aplikacjach w środowisku produkcyjnym, w których możesz pracować z zespołem projektowym, dimens.xmlz łatwością pozwolisz na częstszą zmianę tych wartości.

   dimens.xml

```
<resources>
  <dimen name="min_text_height">48dp</dimen>
</resources>
```

   Zaktualizujesz styles.xmlplik do użycia @dimen/min_text_heightzamiast 48dpbezpośrednio.

```
...
<style name="Widget.TipTime.TextView" parent="Widget.MaterialComponents.TextView">
   <item name="android:minHeight">@dimen/min_text_height</item>
   <item name="android:gravity">center_vertical</item>
   <item name="android:textAppearance">?attr/textAppearanceBody1</item>
</style>
...
```

Dodaj te style do swoich motywów

Być może zauważyłeś, że nie zastosowałeś jeszcze nowych RadioButtoni Switchstylów do odpowiednich widżetów. Powodem jest to, że będziesz używać atrybutów motywu do ustawiania motywu radioButtonStylei switchStylew aplikacji. Wróćmy do tematu.

Motyw to zbiór nazwanych zasobów (zwanych atrybutami motywu), do których można się później odwoływać w stylach, układach itp. Motyw można określić dla całej hierarchii aplikacji, działania lub widoków — nie tylko dla jednej osoby View. Wcześniej modyfikowałeś motyw aplikacji, themes.xmlustawiając atrybuty motywu, takie jak colorPrimaryi colorSecondary, które są używane w całej aplikacji i jej składnikach.

radioButtonStylei switchStylesą innymi atrybutami motywu, które możesz ustawić. Zasoby stylów, które udostępniasz dla tych atrybutów motywu, zostaną zastosowane do każdego przycisku opcji i każdego przełącznika w hierarchii widoków, do którego ma zastosowanie motyw.

Istnieje również atrybut motywu, w textInputStylektórym określony zasób stylu zostanie zastosowany do wszystkich pól wprowadzania tekstu w aplikacji. Aby TextInputLayoutwygląd wyglądał jak obrysowane pole tekstowe (jak pokazano w wytycznych Material Design), istnieje OutlinedBoxstyl zdefiniowany w bibliotece MDC jako Widget.MaterialComponents.TextInputLayout.OutlinedBox. To jest styl, którego będziesz używać.

1. Zmodyfikuj themes.xmlplik tak, aby motyw nawiązywał do żądanych stylów. Ustawienie atrybutu motywu odbywa się w taki sam sposób, w jaki zadeklarowałeś atrybuty colorPrimaryi colorSecondarymotywu we wcześniejszym laboratorium programowania. Jednak tym razem odpowiednie atrybuty motywu to textInputStyle, radioButtonStylei switchStyle. Będziesz używać stylów, które utworzyłeś wcześniej dla RadioButtoni Switchwraz ze stylem dla OutlinedBoxpola tekstowego Materiał.

   Skopiuj następujące elementy do res/values/themes.xmltagu stylu dla motywu aplikacji.

```
<item
name="textInputStyle">@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox</item>
<item name="radioButtonStyle">@style/Widget.TipTime.CompoundButton.RadioButton</item>
<item name="switchStyle">@style/Widget.TipTime.CompoundButton.Switch</item>
```

2. res/values/themes.xmlTak powinien wyglądać twój plik. Jeśli chcesz, możesz dodawać komentarze w pliku XML (oznaczone znakami <!-- i -->).

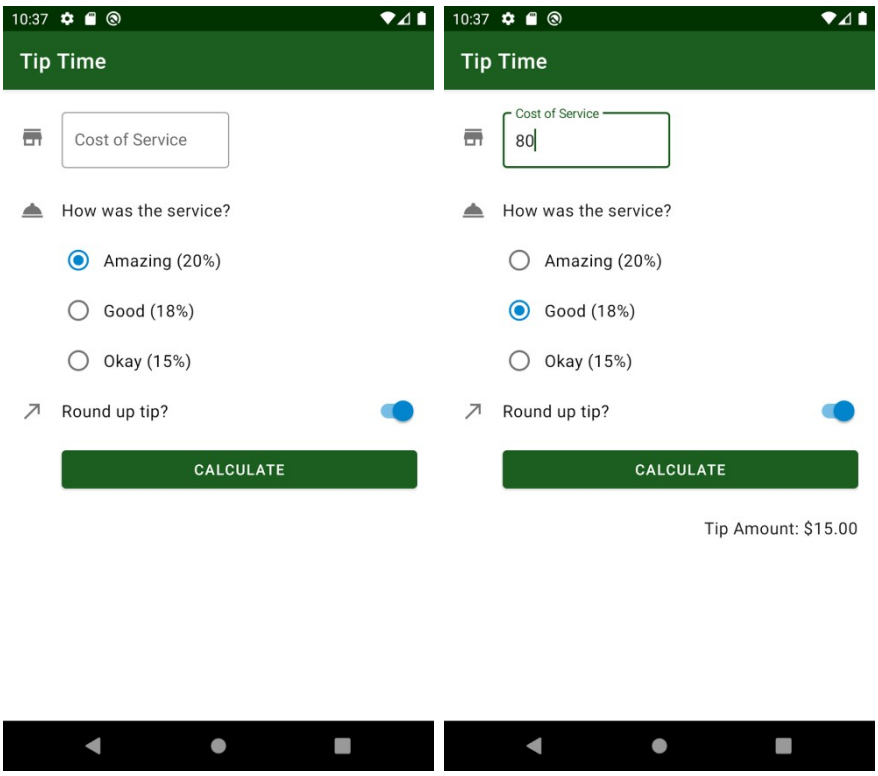
```
<resources xmlns:tools="http://schemas.android.com/tools">

  <!-- Base application theme. -->
  <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    ...
    <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
    <!-- Text input fields -->
    <item
name="textInputStyle">@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox</item>
```

```
    <!-- Radio buttons -->
    <item
name="radioButtonStyle">@style/Widget.TipTime.CompoundButton.RadioButton</item>
    <!-- Switches -->
    <item name="switchStyle">@style/Widget.TipTime.CompoundButton.Switch</item>
  </style>

</resources>
```

3. Pamiętaj, aby wprowadzić te same zmiany w ciemnym motywie w **pliku themes.xml (noc)** . Twój `res/values-night/themes.xml`plik powinien wyglądać tak.

```
<resources xmlns:tools="http://schemas.android.com/tools">

  <!-- Application theme for dark theme. -->
  <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    ...
    <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
    <!-- Text input fields -->
    <item
name="textInputStyle">@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox</item>
    <!-- For radio buttons -->
    <item
name="radioButtonStyle">@style/Widget.TipTime.CompoundButton.RadioButton</item>
    <!-- For switches -->
    <item name="switchStyle">@style/Widget.TipTime.CompoundButton.Switch</item>
  </style>

</resources>
```

4. Uruchom aplikację i zobacz zmiany. Styl `OutlinedBox`wygląda znacznie lepiej dla pola tekstowego, a cały tekst wygląda teraz spójnie!

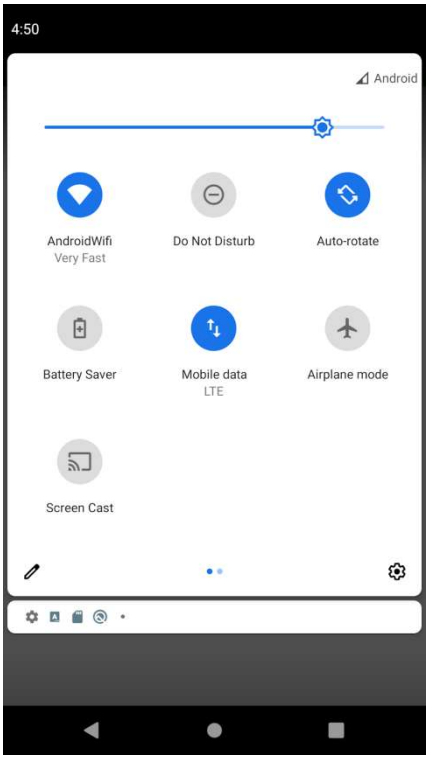


6. Popraw wrażenia użytkownika

W miarę zbliżania się do ukończenia aplikacji należy ją przetestować nie tylko z oczekiwanym przepływem pracy, ale także w innych scenariuszach użytkownika. Może się okazać, że niektóre drobne zmiany w kodzie mogą w znacznym stopniu poprawić wrażenia użytkownika.

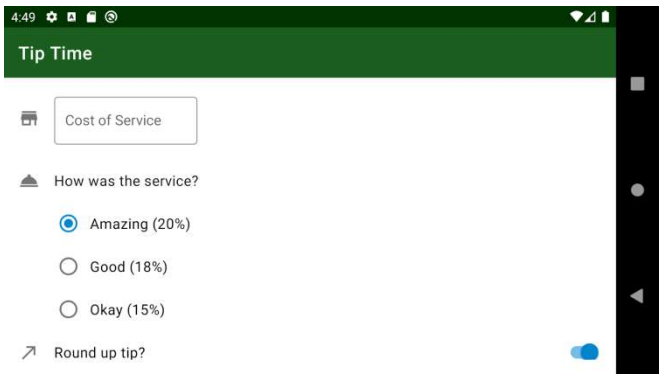
Obracanie urządzenia

1. Obróć urządzenie do trybu poziomego. Może być konieczne wcześniejsze włączenie ustawienia **automatycznego obracania** . (Znajduje się to w **Szybkich ustawieniach** urządzenia lub w Ustawieniach > Wyświetlacz > Zaawansowane > Opcja **automatycznego obracania ekranu).**



W emulatorze możesz następnie użyć opcji emulatora (znajdujących się w prawym górnym rogu obok urządzenia), aby obrócić ekran w prawo lub w lewo.

2. Zauważysz, że niektóre elementy interfejsu użytkownika, w tym przycisk **Oblicz** , zostaną obcięte. To wyraźnie uniemożliwia korzystanie z aplikacji!



3. Aby rozwiązać ten błąd, dodaj `ScrollView`. `ConstraintLayout`Twój XML będzie wyglądał mniej więcej tak.

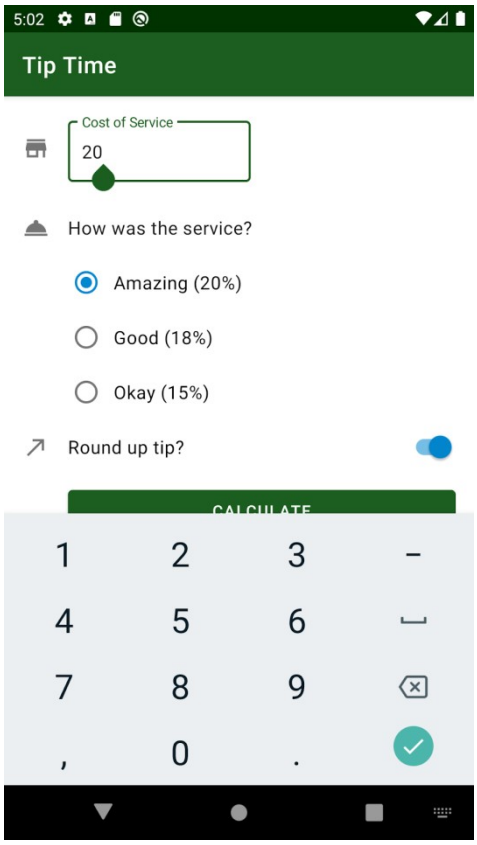
```
<ScrollView
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_height="match_parent"
    android:layout_width="match_parent">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp"
        tools:context=".MainActivity">

        ...
    </ConstraintLayout>

</ScrollView>
```

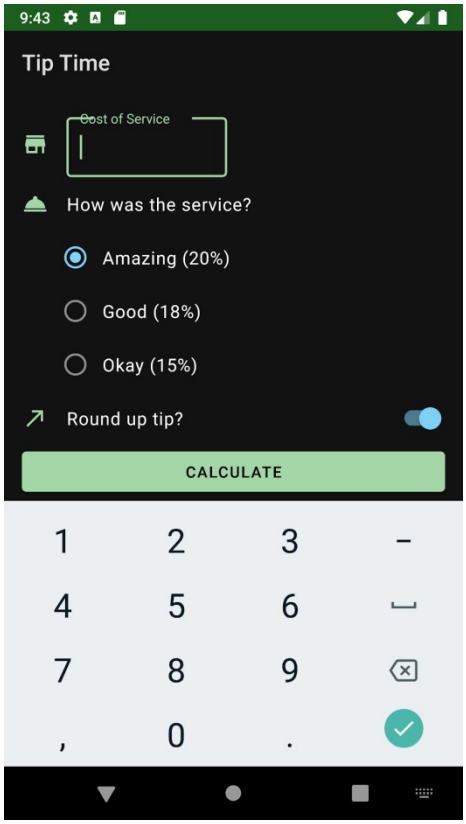
4. Uruchom i ponownie przetestuj aplikację. Po obróceniu urządzenia do trybu poziomego powinno być możliwe przewinięcie interfejsu użytkownika, aby uzyskać dostęp do przycisku obliczania i zobaczyć wynik napiwku. Ta poprawka jest przydatna nie tylko w trybie poziomym, ale także w przypadku innych urządzeń z Androidem, które mogą mieć różne wymiary. Teraz niezależnie od wielkości ekranu urządzenia, użytkownik może przewijać układ.

Ukryj klawiaturę po naciśnięciu klawisza Enter

Być może zauważyłeś, że po wpisaniu kosztu usługi klawiatura nadal działa. Nieco kłopotliwe jest ręczne ukrywanie klawiatury za każdym razem, aby uzyskać lepszy dostęp do przycisku obliczania. Zamiast tego spraw, aby klawiatura automatycznie ukrywała się po naciśnięciu klawisza Enter.



W polu tekstowym możesz zdefiniować detektor klawiszy, który będzie reagował na zdarzenia po dotknięciu określonych klawiszy. Każda możliwa opcja wprowadzania na klawiaturze ma powiązany kod klucza, w tym `Enter`klucz. Pamiętaj, że klawiatura ekranowa jest również nazywana klawiaturą miękką, w przeciwieństwie do klawiatury fizycznej.



W tym zadaniu skonfiguruj detektor klawiszy w polu tekstowym, aby nasłuchiwał `Enter`naciskania klawisza. Po wykryciu tego zdarzenia ukryj klawiaturę.

1. Skopiuj i wklej tę metodę pomocniczą do swojej `MainActivity`klasy. Możesz go wstawić tuż przed nawiasem zamykającym `MainActivity`klasy. Jest `handleKeyEvent()`to prywatna funkcja pomocnicza, która ukrywa klawiaturę ekranową, jeśli `keyCode`parametr wejściowy jest równy `KeyEvent.KEYCODE_ENTER`. InputMethodManager kontroluje, czy klawiatura programowa jest pokazywana, ukryta i pozwala użytkownikowi wybrać, która klawiatura programowa jest wyświetlana. Metoda zwraca true, jeśli zdarzenie klucza zostało obsłużone, aw przeciwnym razie zwraca false.

`MainActivity.kt`

```
private fun handleKeyEvent(view: View, keyCode: Int): Boolean {
  if (keyCode == KeyEvent.KEYCODE_ENTER) {
     // Hide the keyboard
     val inputMethodManager =
        getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
     inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
     return true
  }
  return false
}
```

2. Teraz dołącz słuchacza klucza do TextInputEditTextwidżetu. Pamiętaj, że możesz uzyskać dostęp do TextInputEditTextwidżetu poprzez obiekt powiązania jako binding.costOfServiceEditText.

   Wywołaj setOnKeyListener()metodę na costOfServiceEditTexti przekaż OnKeyListener. Jest to podobne do ustawienia odbiornika kliknięć na przycisku obliczania w aplikacji za pomocą binding.calculateButton.setOnClickListener { calculateTip() }.
   Kod do ustawiania detektora klawiszy w widoku jest nieco bardziej złożony, ale ogólna idea polega na tym, że OnKeyListenerma onKey()metodę, która jest uruchamiana po naciśnięciu klawisza. Metoda onKey()przyjmuje 3 argumenty wejściowe: widok, kod naciśniętego klawisza i zdarzenie klawisza (którego nie będziesz używać, więc możesz go nazwać " _"). Po wywołaniu metody onKey()należy wywołać handleKeyEvent()metodę i przekazać argumenty widoku i kodu klucza. Składnia do zapisania tego to: view, keyCode, _ -> handleKeyEvent(view, keyCode). W rzeczywistości nazywa się to wyrażeniem lambda, ale dowiesz się więcej o lambdach w dalszej części.

   Dodaj kod do ustawienia detektora klucza w polu tekstowym w onCreate()metodzie działania. Dzieje się tak, ponieważ chcesz, aby twój kluczowy odbiornik był dołączony zaraz po utworzeniu układu i zanim użytkownik zacznie wchodzić w interakcję z działaniem.

   MainActivity.kt

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

  setContentView(binding.root)

  binding.calculateButton.setOnClickListener { calculateTip() }

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

3. Sprawdź, czy Twoje nowe zmiany działają. Uruchom aplikację i wprowadź koszt usługi. Naciśnij klawisz Enter na klawiaturze, a miękka klawiatura powinna się ukryć.

Przetestuj swoją aplikację 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 użytkownik może dokończyć przypadek użycia obliczania napiwku w Twojej aplikacji.

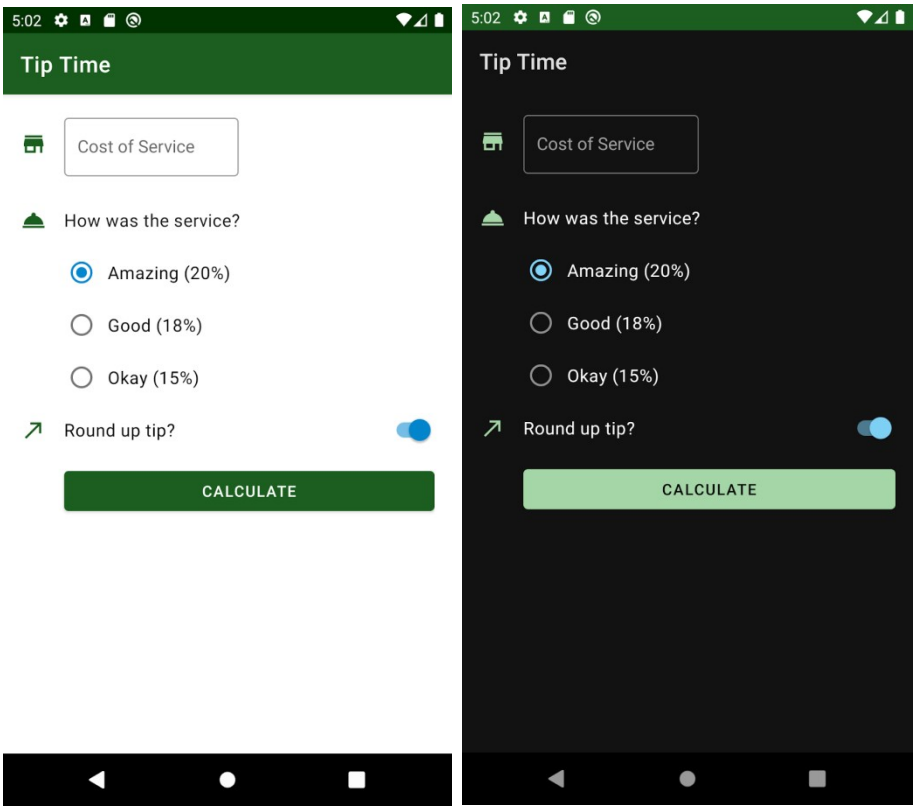
1. Włącz TalkBack na swoim urządzeniu, postępując zgodnie z tymi instrukcjami .
2. Wróć do aplikacji **Porada Czas** .
3. Poznaj swoją aplikację za pomocą Talkback, korzystając z tych instrukcji . Przesuń palcem w prawo, aby poruszać się po elementach ekranu w kolejności, i przesuń w lewo, aby przejść 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 jest w stanie przejść do każdego elementu na ekranie, wprowadzić koszt usługi, zmienić opcje napiwków, obliczyć napiwek i usłyszeć ogłaszany napiwek. Pamiętaj, że dla

ikon nie są podawane żadne komunikaty głosowe, ponieważ oznaczyłeś je jako `importantForAccessibility="no"`.

Aby uzyskać więcej informacji o tym, jak zwiększyć dostępność aplikacji, zapoznaj się z tymi zasadami i tą ścieżką edukacyjną .

(Opcjonalnie) Dostosuj odcień rysunków wektorowych

W tym opcjonalnym zadaniu przyciemnisz ikony na podstawie podstawowego koloru motywu, aby ikony wyglądały inaczej w jasnych i ciemnych motywach (jak pokazano poniżej). Ta zmiana jest miłym dodatkiem do interfejsu użytkownika, aby ikony wyglądały bardziej spójnie z motywem aplikacji.

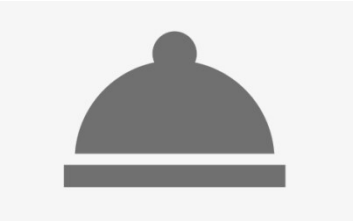


Jak wspomnieliśmy wcześniej, jedną z zalet w `VectorDrawables`porównaniu z obrazami bitmapowymi jest możliwość ich skalowania i przyciemniania. Poniżej mamy XML reprezentujący ikonę dzwonka. Należy zwrócić uwagę na dwa specyficzne atrybuty kolorów: `android:tint`i `android:fillColor`.

`ic_service.xml`

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
  android:width="24dp"
  android:height="24dp"
  android:viewportWidth="24"
  android:viewportHeight="24"
  android:tint="?attr/colorControlNormal">
 <path
    android:fillColor="@android:color/white"
    android:pathData="M2,17h20v2L2,19zM13.84,7.79c0.1,-0.24 0.16,-0.51 0.16,-0.79 0,-1.1 -0.9,-2 -2,-2s-2,0.9 -2,2c0,0.28 0.06,0.55 0.16,0.79C6.25,8.6 3.27,11.93 3,16h18c-0.27,-4.07 -3.25,-7.4 -7.16,-8.21z"/>
```

```
</vector>
```



Jeśli istnieje odcień, zastąpi on wszelkie `fillColor`dyrektywy dotyczące rysowania. W takim przypadku biały kolor jest zastępowany `colorControlNormal`atrybutem motywu. `colorControlNormal`to kolor „normalnego” (stan niezaznaczony/nieaktywowany) widżetu. Obecnie jest to kolor szary.

Jednym z ulepszeń wizualnych, które możemy wprowadzić do aplikacji, jest zabarwienie rysunku na podstawie podstawowego koloru motywu aplikacji. W przypadku jasnego motywu ikona będzie wyglądać jak `@color/green`, natomiast w ciemnym motywie ikona będzie wyglądać jak `@color/green_light`, czyli `?attr/colorPrimary`. Barwienie rysunku w oparciu o podstawowy kolor motywu aplikacji może sprawić, że elementy w układzie będą wyglądać na bardziej ujednolicone i spójne. To również oszczędza nam konieczności duplikowania zestawu ikon dla jasnego i ciemnego motywu. Jest tylko 1 zestaw wektorów do rysowania, a odcień zmieni się w zależności od `colorPrimary`atrybutu motywu.

1. Zmień wartość `android:tint`atrybutu w`ic_service.xml`

```
android:tint="?attr/colorPrimary"
```

W Android Studio ta ikona ma teraz odpowiedni odcień.

Wartość, na którą `colorPrimary`wskazuje atrybut motywu, będzie się różnić w zależności od jasnego i ciemnego motywu.

2. Powtórz to samo, aby zmienić odcień na innych rysunkach wektorowych.

`ic_store.xml`

```
<vector ...
  android:tint="?attr/colorPrimary">
  ...
```

```
</vector>
```

ic_round_up.xml

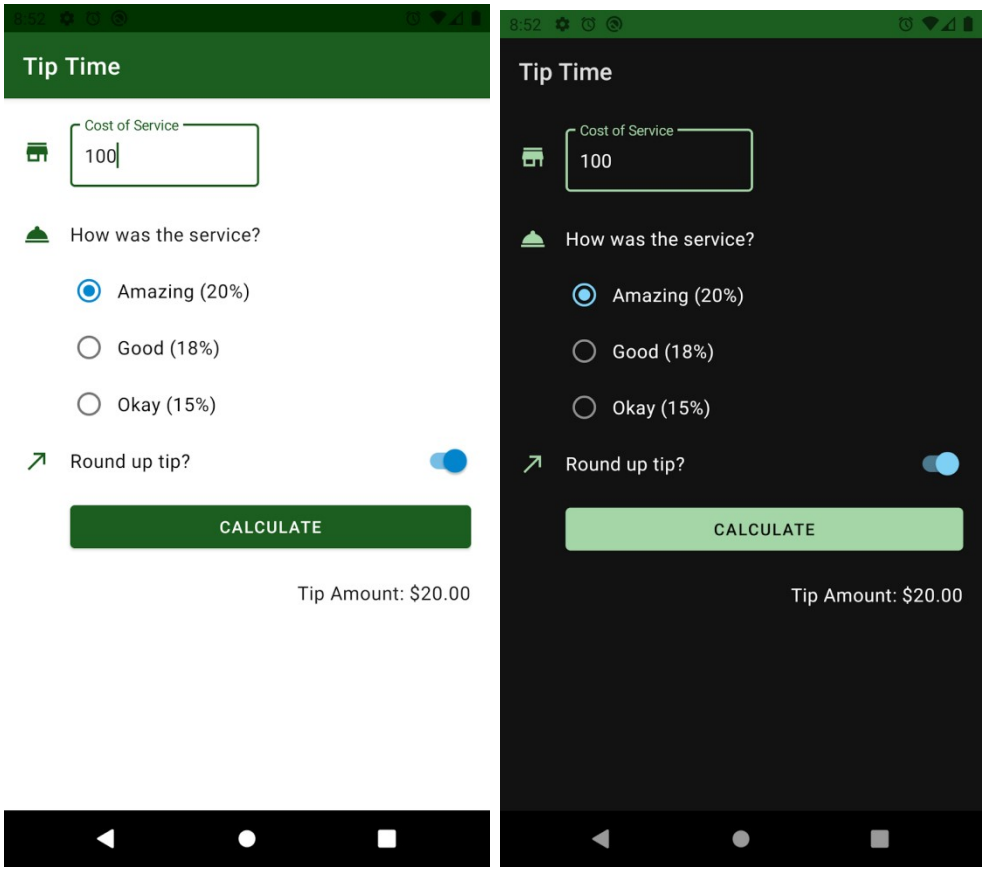
```
<vector ...
  android:tint="?attr/colorPrimary">
  ...
</vector>
```

3. Uruchom aplikację. Sprawdź, czy ikony wyglądają inaczej w jasnych i ciemnych motywach.
4. Na koniec pamiętaj o ponownym sformatowaniu wszystkich plików kodu XML i Kotlin w swojej aplikacji.

Gratulacje, w końcu ukończyłeś aplikację kalkulatora napiwków! Powinieneś być bardzo dumny z tego, co zbudowałeś. Mam nadzieję, że jest to odskocznia do tworzenia jeszcze piękniejszych i bardziej funkcjonalnych aplikacji!

7. Kod rozwiązania

Kod rozwiązania dla tego ćwiczenia programowania znajduje się w repozytorium GitHub wymienionym poniżej.

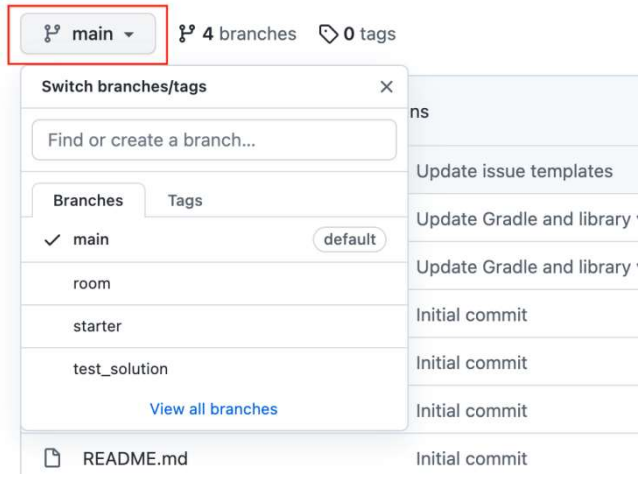


Adres URL kodu rozwiązania: https://github.com/google-developer-training/android-basics-kotlin-tip-calculator-app-solution

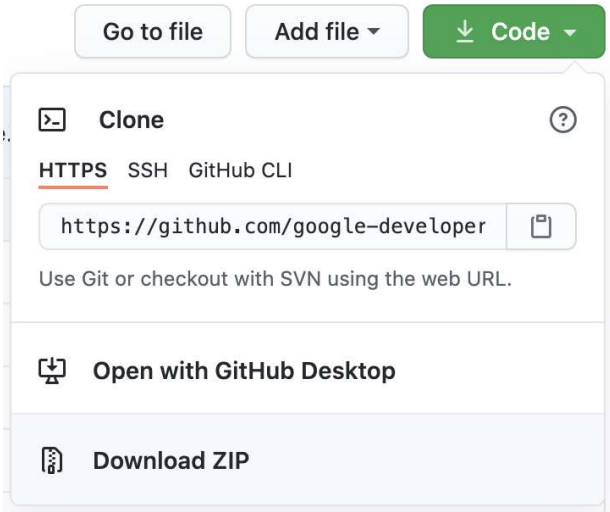
Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

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.

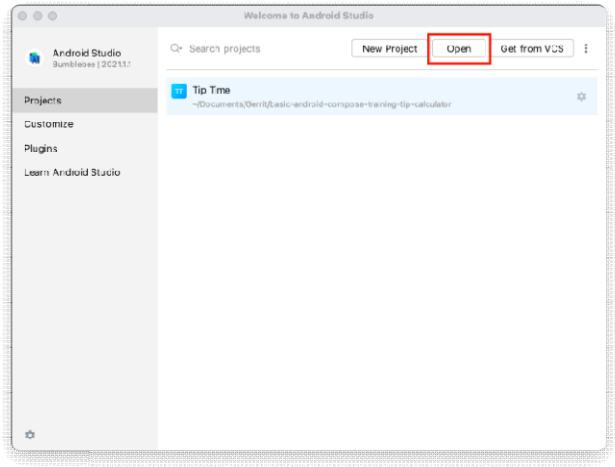


4. W wyskakującym okienku kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na komputerze. Poczekaj na zakończenie pobierania.
5. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
6. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego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

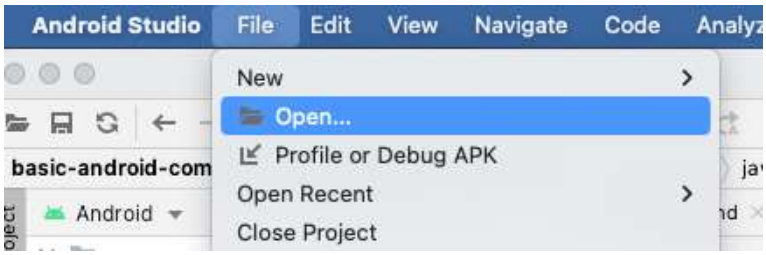
Otwórz projekt w Android Studio

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.

8. Podsumowanie

- Tam, gdzie to możliwe, używaj komponentów Material Design, aby przestrzegać wytycznych dotyczących projektowania materiałów i umożliwić większą personalizację.
- Dodaj ikony, aby dać użytkownikom wizualne wskazówki dotyczące działania części Twojej aplikacji.
- Służy `ConstraintLayout`do pozycjonowania elementów w układzie.
- Przetestuj swoją aplikację pod kątem przypadków brzegowych (np. obracając aplikację w trybie poziomym) i w razie potrzeby wprowadź ulepszenia.
- Skomentuj swój kod, aby pomóc innym osobom, które go czytają, zrozumieć, jakie było Twoje podejście.
- Zmień formatowanie kodu i wyczyść go, aby był jak najbardziej zwięzły.

9. Dowiedz się więcej

- Pola tekstowe
- *Przyciski wyboru: przełączniki*
- Ikony materiałów
- Tematyka typografii
- Powaga
- Udostępnij swoją aplikację na Androida

10. Ćwicz na własną rękę

Uwaga: Praktyki są opcjonalne. Dają ci możliwość przećwiczenia tego, czego nauczyłeś się podczas tego ćwiczenia z programowania.

- W ramach kontynuacji wcześniejszych ćwiczeń z kodowania zaktualizuj aplikację do gotowania konwertera jednostek, aby ściślej przestrzegać wytycznych dotyczących materiałów, korzystając z najlepszych praktyk, których się tutaj nauczyłeś (takich jak używanie komponentów do projektowania materiałów).

Napisz testy oprzyrządowania

1. Zanim zaczniesz

W poprzednim ćwiczeniu z programowania nauczyłeś się tworzyć i uruchamiać testy jednostkowe. To laboratorium kodowania koncentruje się na testach oprzyrządowania. Będziesz miał okazję zobaczyć, jak wyglądają i jak je napisać.

Warunki wstępne

- Utworzyłeś projekt w Android Studio.
- Masz pewne doświadczenie w pisaniu kodu w Android Studio.
- Masz pewne doświadczenie w nawigowaniu projektami w Android Studio.
- Napisałeś proste testy jednostkowe w Android Studio.

Czego się nauczysz

- Jak wyglądają testy oprzyrządowania.
- Jak przeprowadzić testy oprzyrządowania.
- Jak pisać testy oprzyrządowania.

Czego potrzebujesz

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

Pobierz kod startowy do tego ćwiczenia z programowania

W tym laboratorium programowania dodasz testy instrumentacji do aplikacji **Tip Time** z poprzedniego kodu rozwiązania.

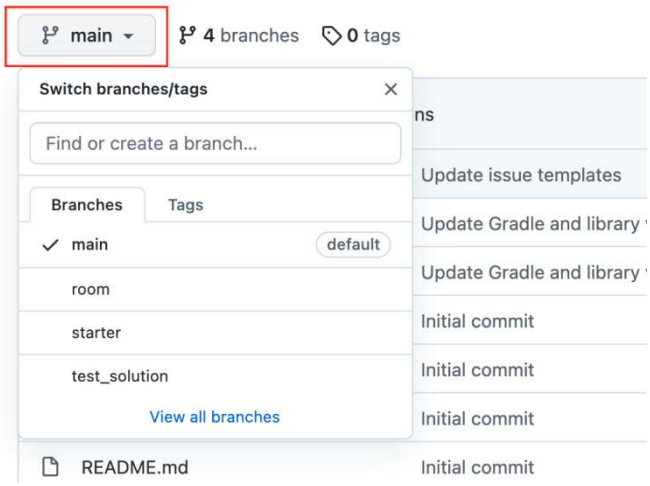
URL kodu startowego: https://github.com/google-developer-training/android-basics-kotlin-tip-calculator-app-solution

Nazwa modułu z kodem startowym:main

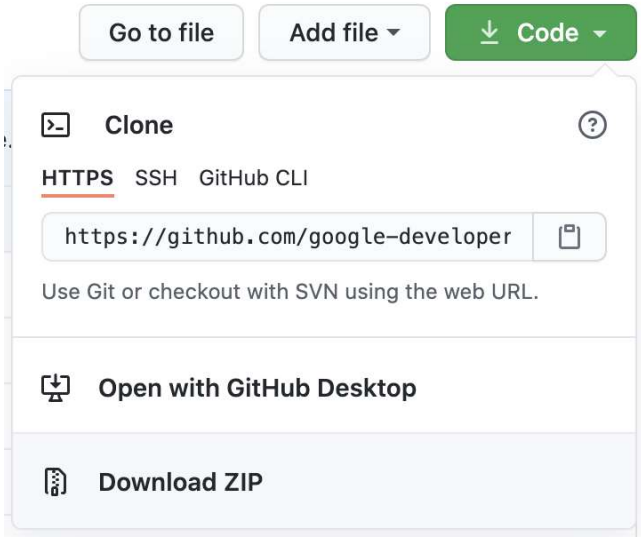
Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

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** .



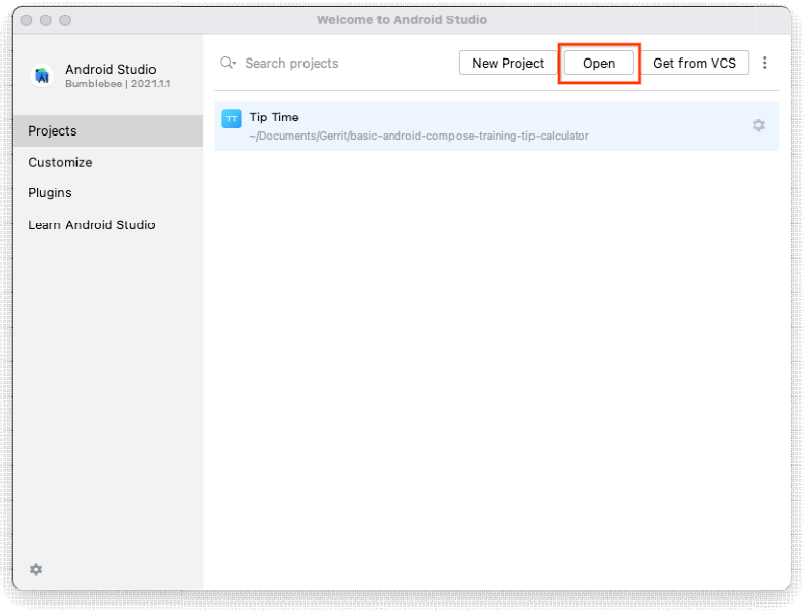
3. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli wyskakujące okienko.



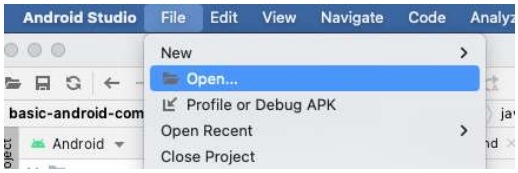
4. W wyskakującym okienku kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na komputerze. Poczekaj na zakończenie pobierania.
5. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
6. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego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

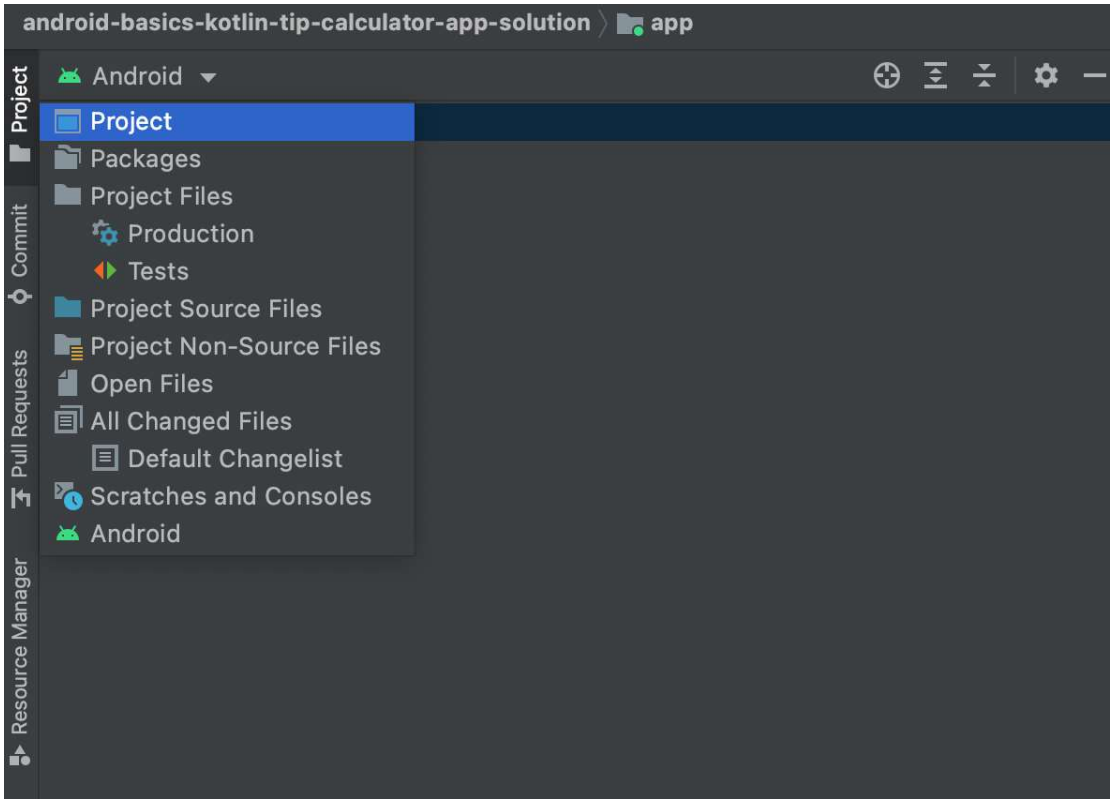
Aplikacja **Czas napiwku** składa się z jednego ekranu, który pokazuje użytkownikowi wprowadzanie tekstu, aby wprowadzić kwotę rachunku, przyciski radiowe do wyboru procentu napiwku oraz przycisk do obliczania napiwku.

3. Utwórz katalog testów oprzyrządowania

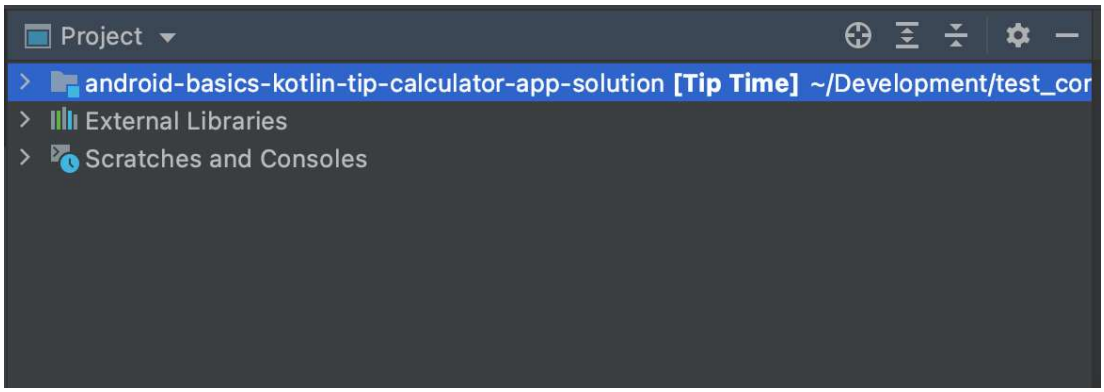
Kod startowy tego laboratorium jest w pełni funkcjonalny, ale brakuje w nim zarówno testów, jak i katalogów testowych. Zanim napiszemy jakiekolwiek testy, musimy dodać katalog do testów instrumentacji. Po pobraniu kodu startowego wykonaj następujące kroki, aby dodać klasę do testów oprzyrządowania.

Uwaga: te katalogi testowe są generowane automatycznie podczas tworzenia nowego projektu.

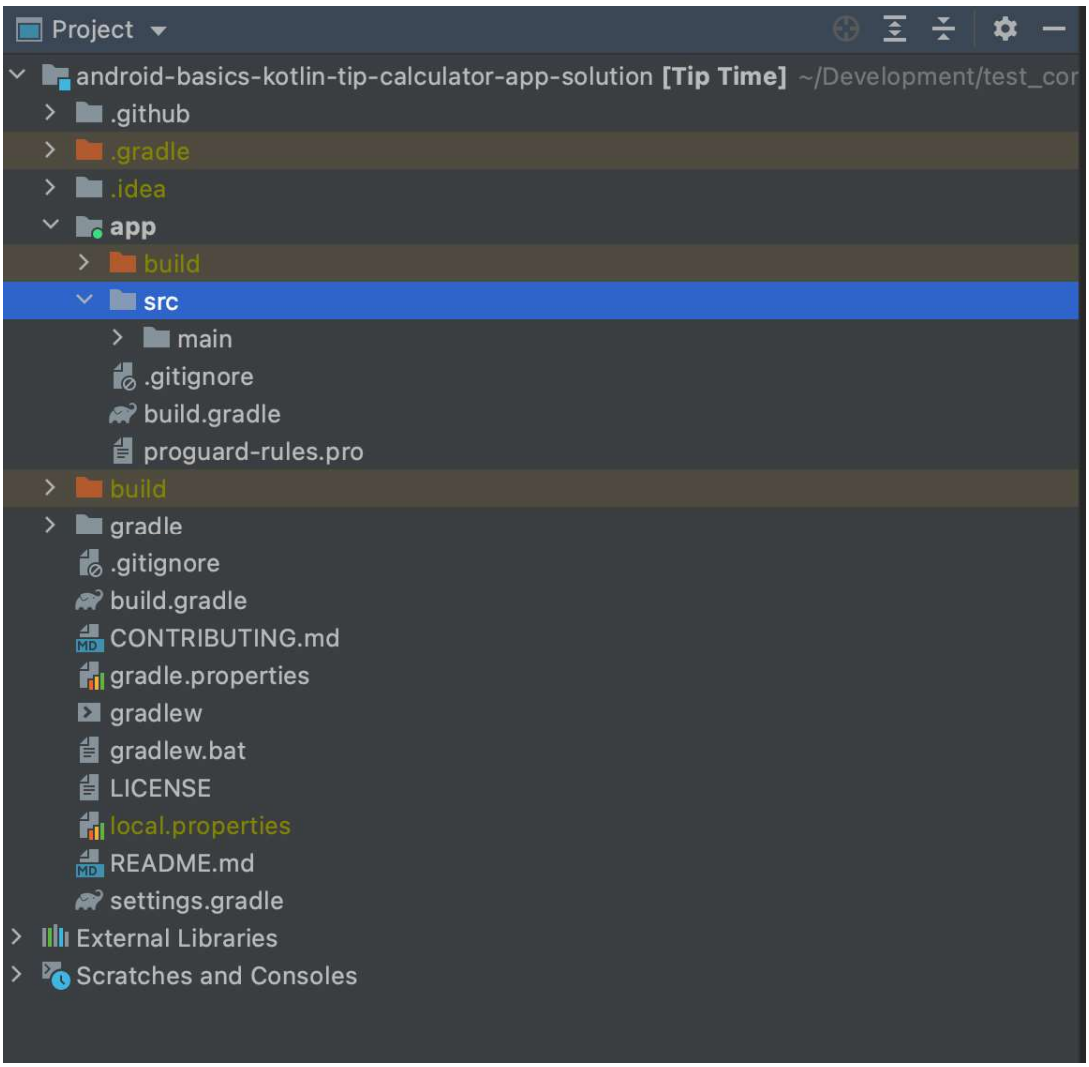
1. Najłatwiej to zrobić, najpierw przełączając się z widoku **Androida** do widoku **Projekt** . W okienku projektu w lewym górnym rogu kliknij listę rozwijaną **Android** i wybierz **Projekt** .



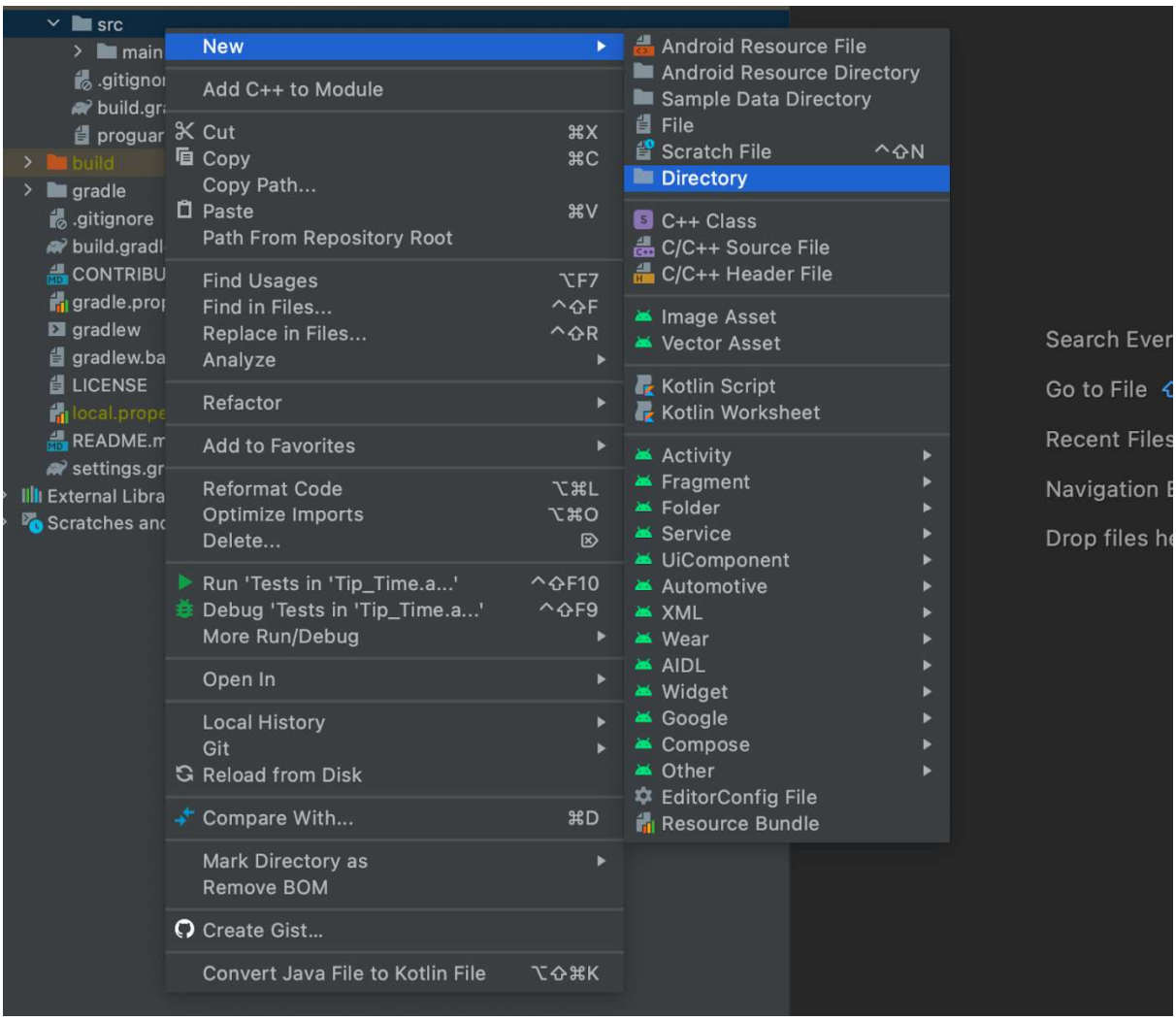
2. Twój widok projektu będzie teraz wyglądał tak:



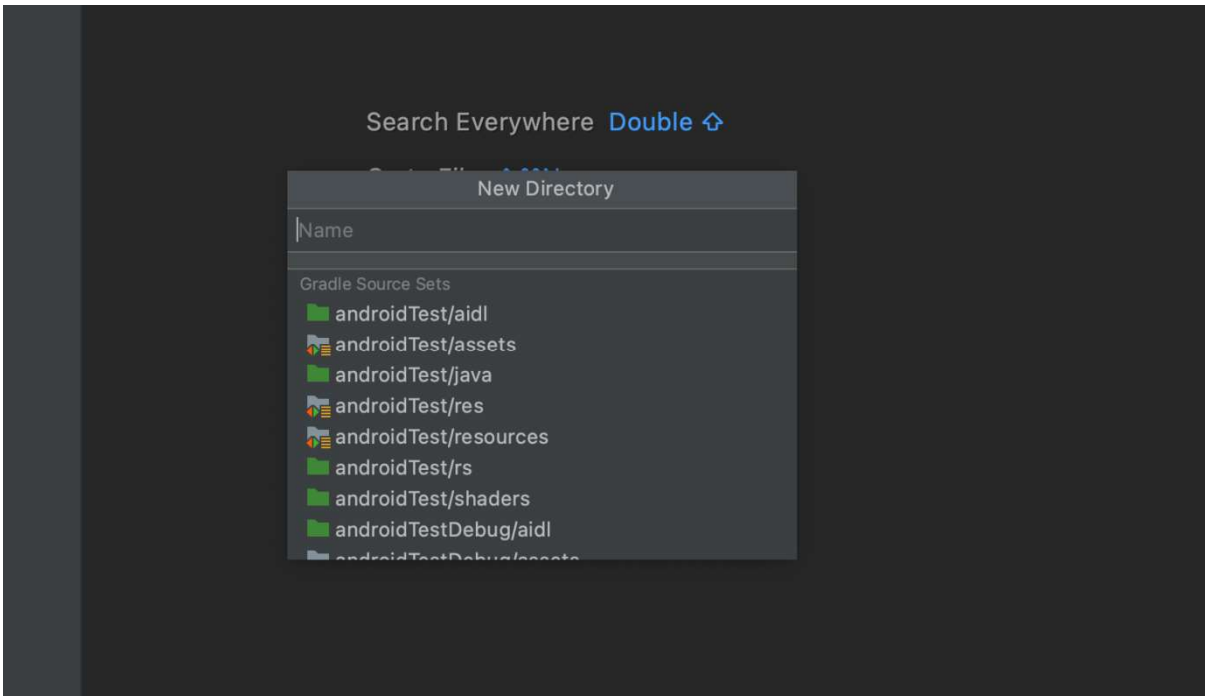
3. Kliknij pierwszą listę rozwijaną, a następnie przejdź do **app -> src** .



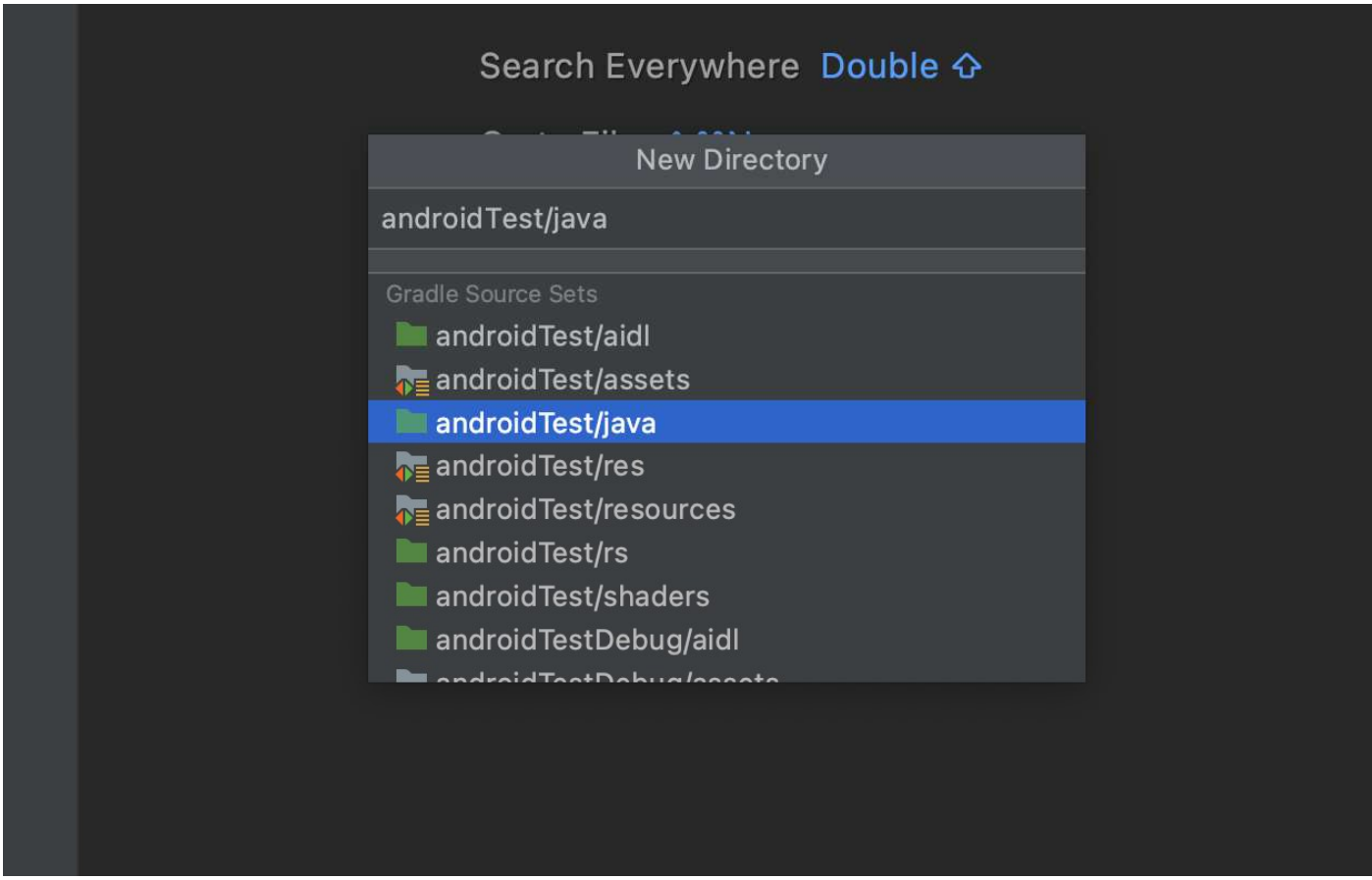
4. Kliknij prawym przyciskiem myszy na **src** i wybierz **Nowy -> Katalog** .



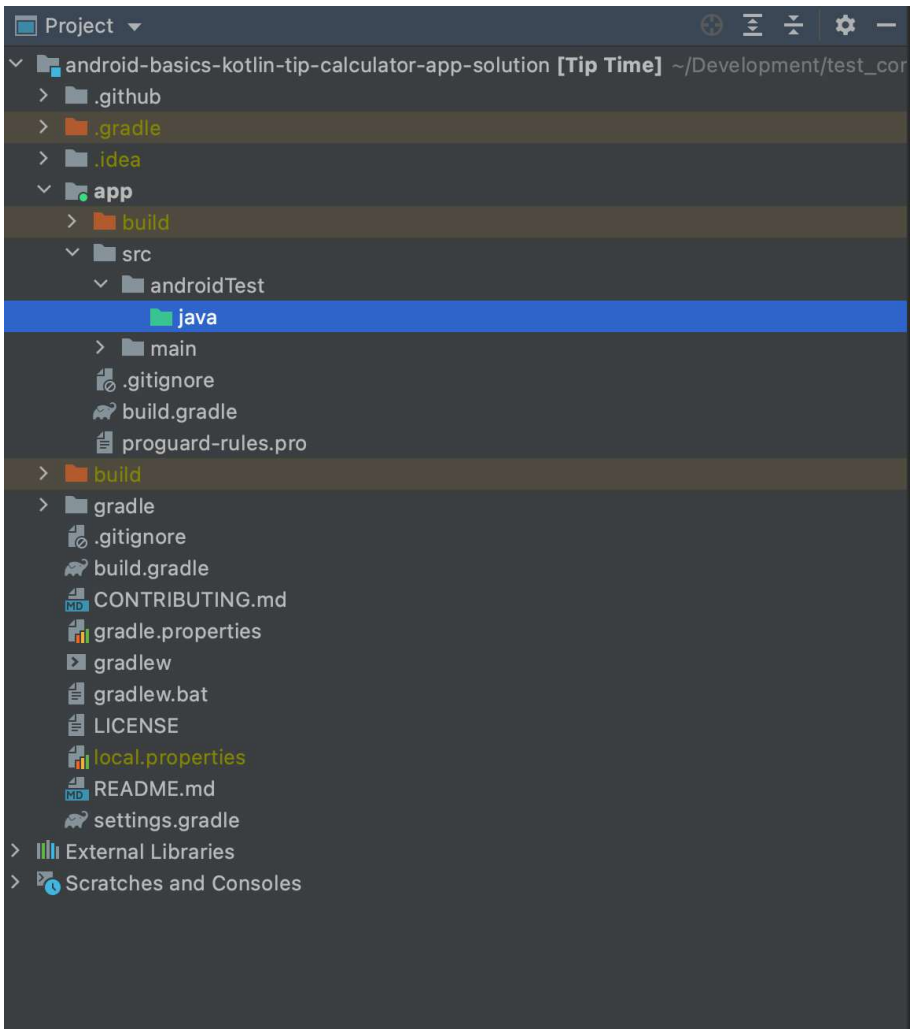
5. Powinieneś zobaczyć to okno:



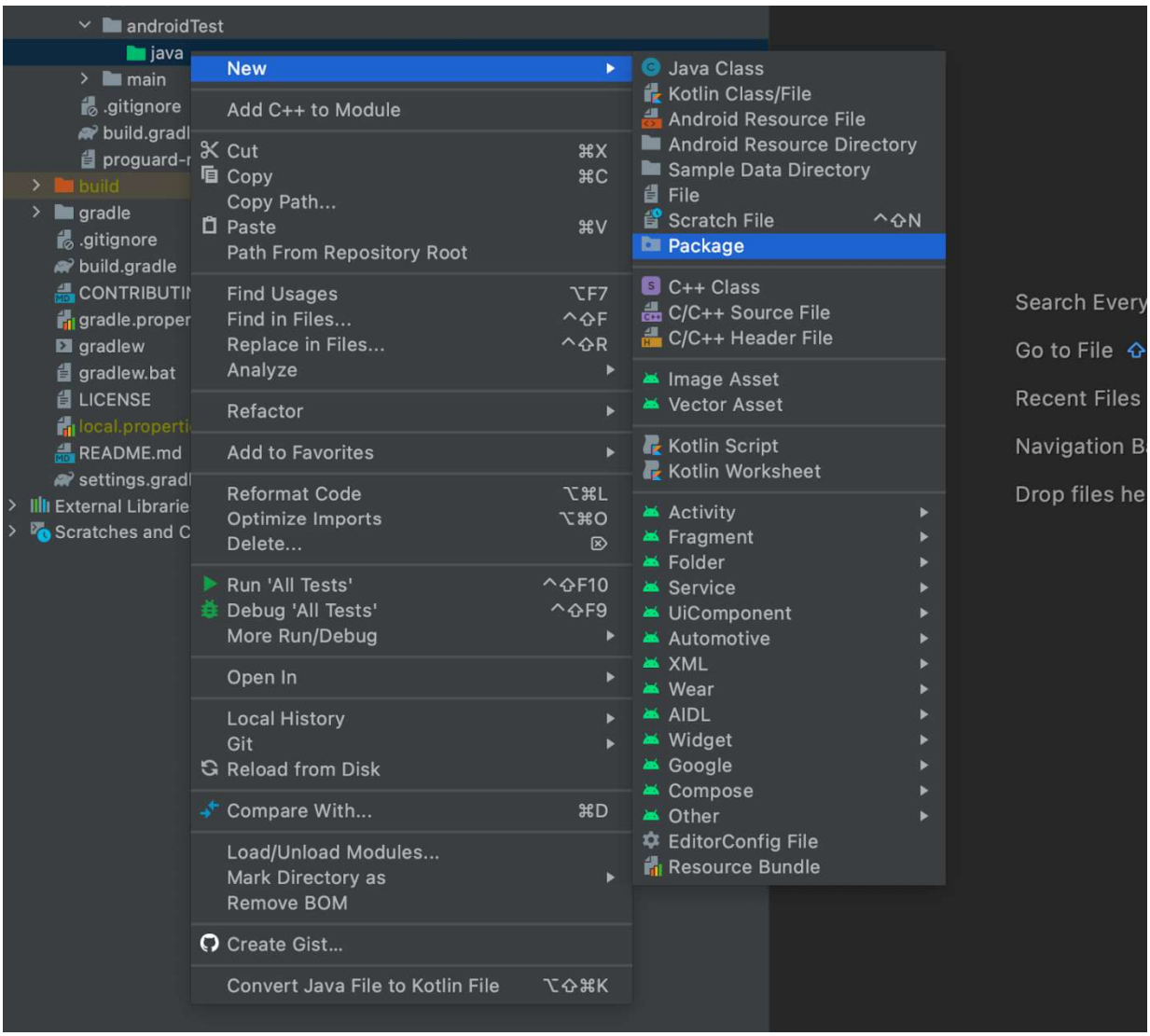
6. Wybierz **androidTest/java** .



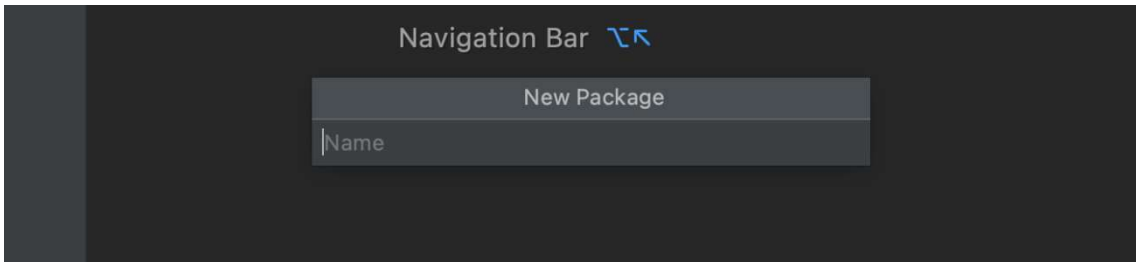
7. Zobaczysz teraz katalog **androidTest** w strukturze projektu.



8. Kliknij prawym przyciskiem myszy katalog **java i wybierz Nowy -> pakiet** .



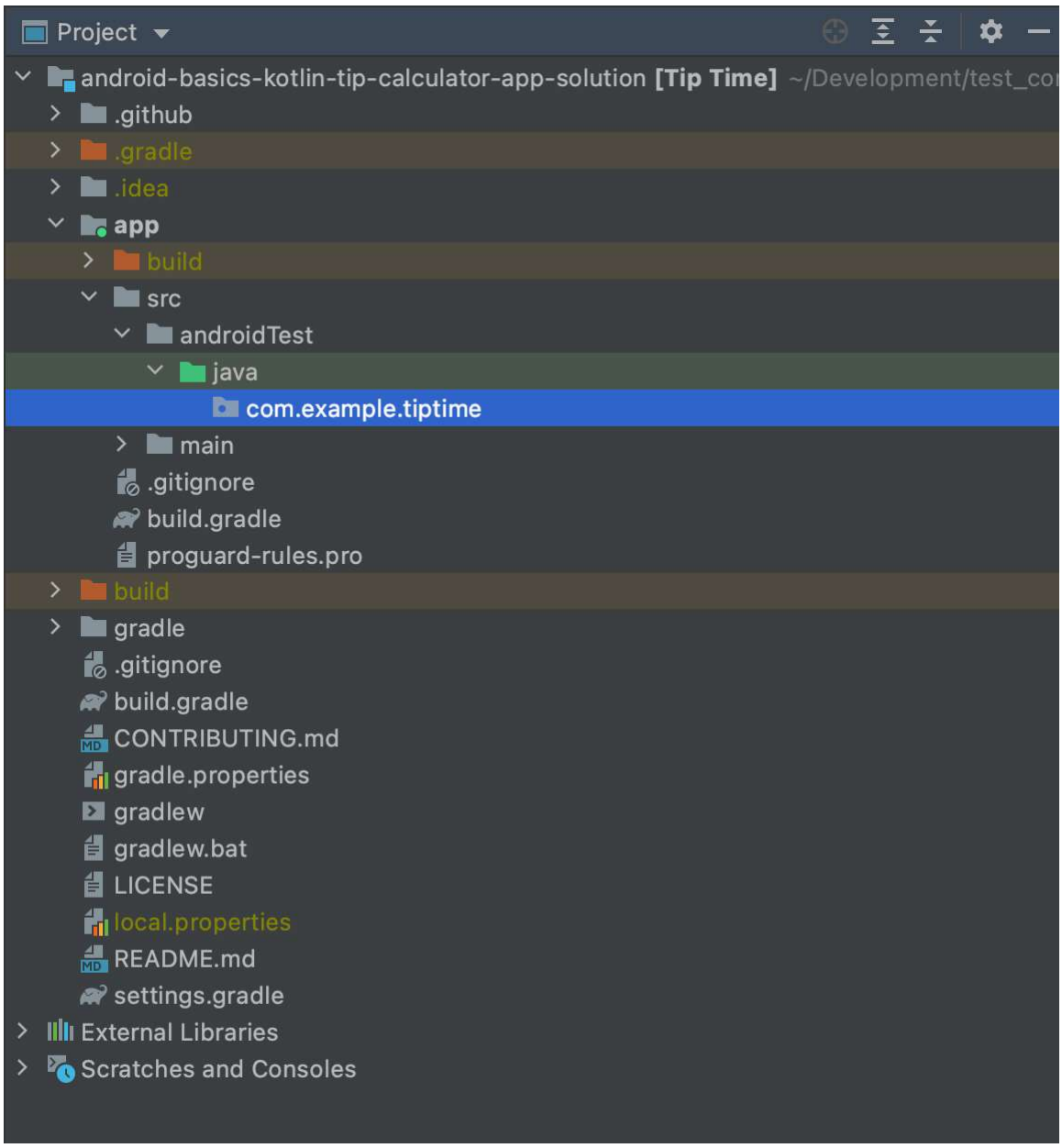
9. Zobaczysz wynikowe okno:



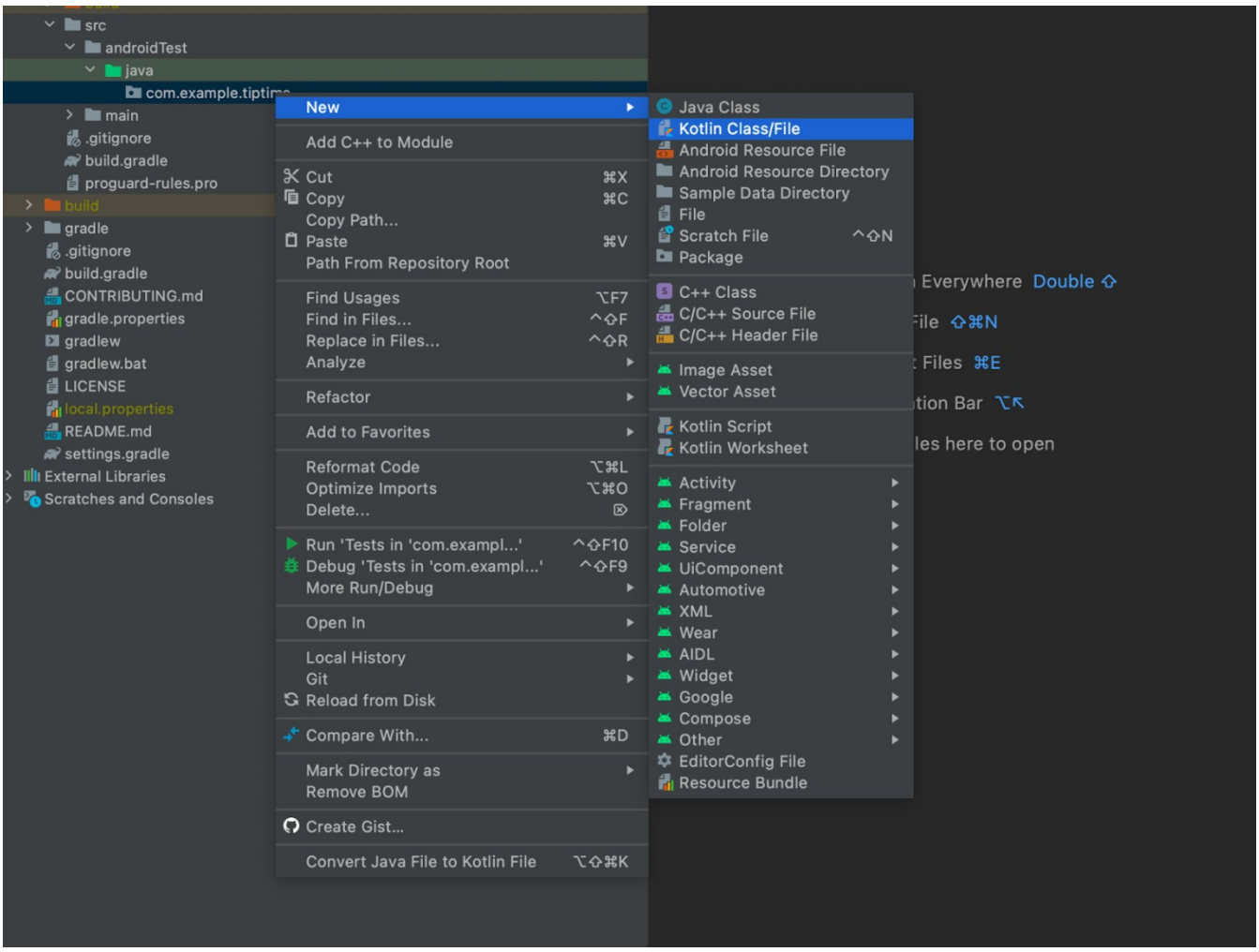
10. W oknie wpisz poniższy tekst i naciśnij **Return** :

    com.example.tiptime

11. Twoje okno projektu powinno wyglądać tak:



12. Na koniec kliknij prawym przyciskiem myszy **com.example.tiptime** i wybierz **Nowy -> Klasa/plik Kotlin** .



13. W wyświetlonym oknie wpisz CalculatorTests, wybierz **Klasa** z listy rozwijanej i naciśnij **Return** .

4. Pisanie pierwszego testu oprzyrządowania

Teraz czas na napisanie testu oprzyrządowania. Poniższe kroki testują funkcjonalność dla 20% napiwku.

1. Otwórz właśnie utworzony plik, powinien wyglądać tak:

```
package com.example.tiptime

class CalculatorTests {
}
```

2. Testy oprzyrządowania wymagają InstrumentationTestRunner , który umożliwia wykonanie testu na urządzeniu lub emulatorze. Istnieje kilka innych biegaczy oprzyrządowania, ale w tym teście użyjemy `AndroidJUnit4`biegacza testowego. Aby określić biegacza testowego, musimy opisać klasę w następujący sposób:

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

```
}
```

Na wszelki wypadek są to importy, których potrzebujesz:

```
import androidx.test.ext.junit.runners.AndroidJUnit4
import org.junit.runner.RunWith
```

Uwaga: Powinieneś być w stanie użyć skrótów klawiszowych, aby automatycznie wygenerować instrukcje importu. W systemie macOS skrótem do automatycznego importowania jest ⌥ (opcja + powrót). W systemie Windows skrótem do automatycznego importowania jest ^ (kontrola + powrót).

Wszystko jest teraz skonfigurowane do rozpoczęcia pisania logiki testowania.

3. Aplikacja **Czas napiwków** składa się z jednej czynności, `MainActivity`. Aby móc wchodzić w interakcję z działaniem, Twój przypadek testowy musi najpierw go uruchomić. Dodaj następujące elementy wewnątrz `CalculatorTests`klasy.

```
@get:Rule()
val activity = ActivityScenarioRule(MainActivity::class.java)
```

`ActivityScenarioRule`pochodzi z biblioteki AndroidX Test. Mówi urządzeniu, aby uruchomiło działanie określone przez programistę. Będziesz musiał dodać adnotację `@get:Rule`, która określa, że kolejna reguła, w tym przypadku uruchamiająca działanie, powinna zostać wykonana przed każdym testem w klasie. Reguły testowe są niezbędnym narzędziem do testowania i ostatecznie nauczysz się pisać własne.

4. Następnie musisz napisać samą logikę testu. Utwórz funkcję, wywołaj ją `calculate_20_percent_tip()`i dodaj `@Test`adnotację.

```
@Test
fun calculate_20_percent_tip() {
}
```

Espresso

Ten kurs wykorzystuje przede wszystkim Espresso do testów oprzyrządowania. Espresso to biblioteka, która jest gotowa do użycia z projektem Androida po utworzeniu w Android Studio i umożliwia interakcję ze składnikami interfejsu użytkownika za pomocą kodu.

W tym momencie mogłeś zauważyć, że Android Studio ma wiele funkcji autouzupełniania. Jednym z trudnych aspektów pracy z Espresso jest to, że metody nie uzupełniają się automatycznie, jeśli biblioteka nie została zaimportowana. Może to utrudnić poruszanie się po dostępnych metodach w Espresso bez badania dokumentacji. Podczas tych lekcji będziemy dostarczać Ci metody potrzebne do ukończenia testów.

Najpierw musisz napisać kod, aby wprowadzić kwotę rachunku w widoku tekstu wejściowego **Koszt usługi .** Przejście do **app -> src -> main -> res -> layout -> activity_main.xml** wskazuje, że identyfikator `TextInputEditText`to `cost_of_service_edit_text`. Skopiuj nazwę ID, której będziemy potrzebować później do testu.

```
<com.google.android.material.textfield.TextInputEditText
    android:id="@+id/cost_of_service_edit_text"
```

Zaimplementuj funkcję testową

Teraz w `calculate_20_percent_tip()` funkcji w klasie testowej możesz napisać logikę testu.

1. Pierwszym krokiem jest znalezienie składnika interfejsu użytkownika do interakcji, w tym przypadku `TextInputEditText`, przy użyciu `onView()` funkcji. Funkcja `onView()` przyjmuje `ViewMatcher` parametr obiektu. A `ViewMatcher` to zasadniczo składnik interfejsu użytkownika, który spełnia określone kryteria, czyli w tym przypadku składnik o identyfikatorze `R.id.cost_of_service_edit_text`.

   Funkcja `withId()` zwraca a `ViewMatcher`, który jest składnikiem interfejsu użytkownika z przekazanym do niego identyfikatorem. `onView()` zwraca a `ViewInteraction`, czyli obiekt, z którym możemy wchodzić w interakcje tak, jakbyśmy sami sterowali urządzeniem. Aby wprowadzić jakiś tekst, dzwonisz `perform()` na `ViewInteraction`. Następnie `perform()` bierze `ViewAction` przedmiot. Istnieje wiele metod, które zwracają a, `ViewAction` ale na razie użyjemy `typeText()` metody. `activity_main.xml` Widzimy, że domyślna opcja napiwków to 20%, więc na razie nie musisz określać, którą opcję napiwków wybrać .

```
onView(withId(R.id.cost_of_service_edit_text))
  .perform(typeText("50.00"))
```

Uwaga: na mniejszych urządzeniach przycisk może być ukryty pod klawiaturą, która została otwarta w celu wpisania tekstu w powyższym kroku. Jeśli używasz mniejszego urządzenia, możesz również połączyć tę metodę z powyższą instrukcją, aby zamknąć klawiaturę:

`.perform(ViewActions.closeSoftKeyboard())`

Po czym cała instrukcja wyglądałaby następująco:

```
onView(withId(R.id.cost_of_service_edit_text))
  .perform(typeText("50.00"))
  .perform(ViewActions.closeSoftKeyboard())
```

2. Tekst jest teraz wprowadzony i test musi kliknąć przycisk **Oblicz** . Kod do tego ma format podobny do tego, którego używaliśmy do wprowadzania tekstu. Składnik interfejsu użytkownika jest inny, dlatego nazwa identyfikatora przekazywana do `withText()` funkcji jest inna. Jednak jedyną różnicą w podejściu jest to, że `ViewAction` jest inne; funkcja `click()` jest używana zamiast `typeText()`.

```
onView(withId(R.id.calculate_button))
  .perform(click())
```

3. Na koniec musisz potwierdzić, że wyświetlana jest właściwa wskazówka. Oczekujemy, że kwota napiwku wyniesie 10,00 USD. W przypadku tego testu upewnij się, że `TextViewz` identyfikatorem `tip_result` zawiera oczekiwaną wartość końcówki w postaci ciągu.

```
onView(withId(R.id.tip_result))
  .check(matches(withText(containsString("$10.00"))))
```

Po wyświetleniu monitu wybierz następujące importy:

```
import androidx.test.espresso.assertion.ViewAssertions.matches
import org.hamcrest.Matchers.containsString
```

W tym przypadku użyłeś innej interakcji o nazwie `check()`, która wymaga `ViewAssertion`. Możesz myśleć o a `ViewAssertion`jako o specjalnym asercji Espresso używanej w komponentach interfejsu użytkownika. Twierdzenie polega na tym, że treść `TextView` dopasowania jest zgodna z tekstem zawierającym ciąg `"$10.00"`.

Przed uruchomieniem testu upewnij się, że importy i kod są poprawne, powinno to wyglądać mniej więcej tak (może być w porządku, jeśli importy są w innej kolejności):

```
package com.example.tiptime

import androidx.test.espresso.Espresso.onView
import androidx.test.espresso.action.ViewActions.click
import androidx.test.espresso.action.ViewActions.typeText
import androidx.test.espresso.assertion.ViewAssertions.matches
import androidx.test.espresso.matcher.ViewMatchers.withId
import androidx.test.espresso.matcher.ViewMatchers.withText
import androidx.test.ext.junit.rules.ActivityScenarioRule
import androidx.test.ext.junit.runners.AndroidJUnit4
import org.hamcrest.Matchers.containsString
import org.junit.Rule
import org.junit.Test
import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)
class CalculatorTests {

  @get:Rule()
  val activity = ActivityScenarioRule(MainActivity::class.java)

  @Test
  fun calculate_20_percent_tip() {
    onView(withId(R.id.cost_of_service_edit_text))
      .perform(typeText("50.00"))

    onView(withId(R.id.calculate_button)).perform(click())

    onView(withId(R.id.tip_result))
      .check(matches(withText(containsString("$10.00"))))
  }
}
```

Jeśli używasz emulatora, upewnij się, że jednocześnie widzisz emulator i okno Android Studio. Uruchom test w taki sam sposób, w jaki uruchamiasz testy jednostkowe (klikając prawym przyciskiem myszy

przycisk z czerwoną/zieloną strzałką po lewej stronie funkcji i wybierając pierwszą opcję) i obserwuj, co się stanie!

Widać, że test działa tak, jakby ktoś wchodził w interakcję z samą aplikacją!

5. Rozwiń swój zestaw testowy

Gratulacje! Masz swój pierwszy test oprzyrządowania do pracy!

Jeśli masz ochotę na wyzwanie, możesz rozszerzyć zestaw testów, dodając funkcje testujące inne wartości procentowe napiwków. Użyj tego samego formatu, co funkcja, którą napisaliśmy powyżej, jedyne zmiany, które musisz wprowadzić, to napisać kod, aby wybrać inną opcję procentową i zmienić wartość przekazaną `containsString()`do uwzględnienia różnych oczekiwanych wyników. Nie zapominaj, że istnieje również opcja zaokrąglania w górę. Możesz przełączyć przełącznik zaokrąglania w górę, znajdując składnik według jego identyfikatora, jak pokazaliśmy za pomocą `onView(withId())`, i klikając go.

6. Kod rozwiązania

Adres URL kodu rozwiązania: https://github.com/google-developer-training/android-basics-kotlin-tip-calculator-app-solution

Nazwa modułu z kodem rozwiązania:

`test_solution`

7. Podsumowanie

- Android Studio generuje niezbędne klasy testowe podczas tworzenia projektu. Jeśli jednak napotkasz projekt, który ich nie posiada, możesz je utworzyć ręcznie.
- Reguły testowe są uruchamiane przed każdym testem w klasie testowej.
- Espresso jest podstawowym elementem testów oprzyrządowania. Umożliwia interakcję z komponentami UI za pomocą kodu.

To była długa lekcja, poklep się po plecach za dobrze wykonaną pracę!

8. Dowiedz się więcej

- Podstawy espresso

Ścieżka 3. Wyświetl przewijalną listę

Utwórz aplikację, która wyświetla przewijaną listę inspirującego tekstu i obrazów, korzystając z widżetu RecyclerView w systemie Android.

Użyj list w Kotlin

1. Zanim zaczniesz

Powszechne jest tworzenie list dla różnych sytuacji w życiu codziennym, takich jak lista rzeczy do zrobienia, lista gości na wydarzenie, lista życzeń lub lista zakupów. W programowaniu bardzo przydatne są również listy. Na przykład w aplikacji może znajdować się lista artykułów z wiadomościami, piosenek, wydarzeń w kalendarzu lub postów w mediach społecznościowych.

Nauka tworzenia i używania list jest ważną koncepcją programistyczną, którą należy dodać do swojego zestawu narzędzi i umożliwi tworzenie bardziej wyrafinowanych aplikacji.

W tym laboratorium kodowania użyjesz Placu zabaw Kotlin, aby zapoznać się z listami w Kotlin i stworzyć program do zamawiania różnych odmian zupy z makaronem. Czy jesteś już głodny?

Warunki wstępne

- Zaznajomiony z wykorzystaniem Kotlin Playground do tworzenia i edycji programów Kotlin.
- Zapoznanie z podstawowymi koncepcjami programowania Kotlin z części 1 kursu Android Basics in Kotlin: `main()`funkcja, argumenty funkcji i wartości zwracane, zmienne, typy danych i operacje, a także instrukcje przepływu sterowania.
- Potrafi zdefiniować klasę Kotlin, utworzyć z niej instancję obiektu i uzyskać dostęp do jej właściwości i metod.
- Potrafi tworzyć podklasy i rozumieć, w jaki sposób dziedziczą po sobie.

Czego się nauczysz

- Jak tworzyć i używać list w Kotlin
- Różnica między `Lista` `MutableList`, a kiedy użyć każdego z nich
- Jak iterować po wszystkich elementach listy i wykonywać działania na każdym elemencie.

Co zbudujesz

- Będziesz eksperymentować z listami i operacjami na listach w Kotlin Playground .
- Stworzysz program do zamawiania jedzenia, który wykorzystuje listy na Placu Zabaw Kotlin .
- Twój program będzie mógł stworzyć zamówienie, dodać do niego makaron i warzywa, a następnie obliczyć całkowity koszt zamówienia.

Czego potrzebujesz

- Komputer z dostępem do Internetu umożliwiający dostęp do Placu Zabaw Kotlin .

2. Wprowadzenie do list

We wcześniejszych ćwiczeniach z programowania poznałeś podstawowe typy danych w Kotlin, takie jak `Int`, `Double`, `Boolean`i `String`. Pozwalają na przechowywanie określonego typu wartości w zmiennej. Ale co, jeśli chcesz przechowywać więcej niż jedną wartość? `List`W tym przypadku przydatne jest posiadanie typu danych.

Lista to zbiór elementów o określonej kolejności. W Kotlinie istnieją dwa rodzaje list:

- Lista tylko do odczytu: `List`nie można jej modyfikować po utworzeniu.
- Lista mutowalna: `MutableList`można ją modyfikować po jej utworzeniu, co oznacza, że można dodawać, usuwać lub aktualizować jej elementy.

Używając `List`lub `MutableList`, musisz określić typ elementu, który może zawierać. Na przykład `List<Int>`przechowuje listę liczb całkowitych i `List<String>`przechowuje listę ciągów. Jeśli zdefiniujesz `Car`klasę w swoim programie, możesz mieć klasę, `List<Car>`która przechowuje listę `Car`instancji obiektów.

Najlepszym sposobem na zrozumienie list jest wypróbowanie ich.

Stwórz listę

1. Otwórz Plac zabaw Kotlin i usuń dostarczony istniejący kod.
2. Dodaj pustą `main()`funkcję. Wszystkie poniższe kroki kodu zostaną umieszczone w tej `main()`funkcji.

```
fun main() {


}
```

3. Wewnątrz `main()`stwórz zmienną o nazwie `numbers`type, `List<Int>`ponieważ będzie ona zawierać listę liczb całkowitych tylko do odczytu. Utwórz nową `List`za pomocą funkcji biblioteki standardowej Kotlin `listOf()`i przekaż elementy listy jako argumenty oddzielone przecinkami. `listOf(1, 2, 3, 4, 5, 6)`zwraca listę liczb całkowitych tylko do odczytu od 1 do 6.

```
val numbers: List<Int> = listOf(1, 2, 3, 4, 5, 6)
```

4. Jeśli typ zmiennej można odgadnąć (lub wywnioskować) na podstawie wartości po prawej stronie operatora przypisania (=), można pominąć typ danych zmiennej. Dlatego możesz skrócić ten wiersz kodu do następującego:

```
val numbers = listOf(1, 2, 3, 4, 5, 6)
```

5. Użyj `println()`, aby wydrukować `numbers`listę.

```
println("List: $numbers")
```

Pamiętaj, że umieszczenie $ w ciągu oznacza, że następuje wyrażenie, które zostanie ocenione i dodane do tego ciągu (zobacz szablony ciągów). Ten wiersz kodu można również zapisać jako`println("List: " + numbers)`.

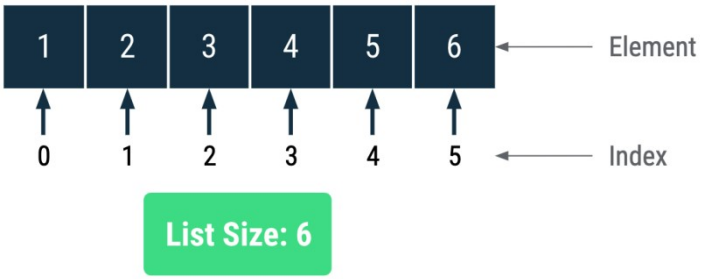
6. Pobierz rozmiar listy za pomocą `numbers.size`właściwości i wydrukuj go również.

```
println("Size: ${numbers.size}")
```

7. Uruchom swój program. Wynikiem jest lista wszystkich elementów listy oraz rozmiar listy. Zwróć uwagę na nawiasy []wskazujące, że jest to List. Wewnątrz nawiasów znajdują się elementy numbers, oddzielone przecinkami. Zauważ też, że elementy są w tej samej kolejności, w jakiej je utworzyłeś.

   List: [1, 2, 3, 4, 5, 6]
   Size: 6
   Elementy listy dostępu

   Funkcjonalność specyficzna dla list polega na tym, że możesz uzyskać dostęp do każdego elementu listy za pomocą jego indeksu, który jest liczbą całkowitą reprezentującą pozycję. To jest diagram numbersutworzonej przez nas listy, pokazujący każdy element i odpowiadający mu indeks.



   Indeks jest w rzeczywistości przesunięciem od pierwszego elementu. Na przykład, gdy mówisz, list[2]że nie pytasz o drugi element listy, ale o element, który jest przesunięty o 2 pozycje od pierwszego elementu. Stąd list[0]pierwszy element (przesunięcie zero), list[1]drugi element (przesunięcie 1), list[2]trzeci element (przesunięcie 2) i tak dalej.

   Dodaj następujący kod po istniejącym kodzie w main()funkcji. Uruchom kod po każdym kroku, aby sprawdzić, czy dane wyjściowe są zgodne z oczekiwaniami.

1. Wydrukuj pierwszy element listy pod indeksem 0. Możesz wywołać get()funkcję z żądanym indeksem jako numbers.get(0)lub możesz użyć skróconej składni z nawiasami kwadratowymi wokół indeksu jako numbers[0].

   ```
   println("First element: ${numbers[0]}")
   ```

2. Następnie wydrukuj drugi element listy pod indeksem 1.

   ```
   println("Second element: ${numbers[1]}")
   ```

   Prawidłowe wartości indeksów ("indeksy") listy idą od 0 do ostatniego indeksu, który jest rozmiarem listy minus 1. Oznacza to, że dla twojej numberslisty indeksy biegną od 0 do 5.

3. Wydrukuj ostatni element listy, używając numbers.size - 1do obliczenia jego indeksu, którym powinno być 5. Dostęp do elementu pod 5 indeksem powinien zostać zwrócony 6jako wyjście.

   ```
   println("Last index: ${numbers.size - 1}")
   println("Last element: ${numbers[numbers.size - 1]}")
   ```

4. Kotlin również wspiera first()i last()operuje na liście. Spróbuj zadzwonić numbers.first()i numbers.last()zobacz wyniki.

```
println("First: ${numbers.first()}")
println("Last: ${numbers.last()}")
```

Zauważysz, że `numbers.first()`zwraca pierwszy element listy i `numbers.last()`zwraca ostatni element listy.

5. Inną przydatną operacją na liście jest `contains()`metoda sprawdzania, czy dany element znajduje się na liście. Na przykład, jeśli masz listę nazwisk pracowników w firmie, możesz użyć `contains()`metody, aby dowiedzieć się, czy dane nazwisko znajduje się na liście.

   Na swojej `numbers`liście wywołaj `contains()`metodę z jedną z liczb całkowitych znajdujących się na liście. `numbers.contains(4)`zwróci wartość `true`. Następnie wywołaj `contains()`metodę z liczbą całkowitą, której nie ma na Twojej liście. `numbers.contains(7)`wróci `false`.

```
println("Contains 4? ${numbers.contains(4)}")
println("Contains 7? ${numbers.contains(7)}")
```

6. Twój ukończony kod powinien wyglądać tak. Komentarze są opcjonalne.

```
fun main() {
  val numbers = listOf(1, 2, 3, 4, 5, 6)
  println("List: $numbers")
  println("Size: ${numbers.size}")

  // Access elements of the list
  println("First element: ${numbers[0]}")
  println("Second element: ${numbers[1]}")
  println("Last index: ${numbers.size - 1}")
  println("Last element: ${numbers[numbers.size - 1]}")
  println("First: ${numbers.first()}")
  println("Last: ${numbers.last()}")

  // Use the contains() method
  println("Contains 4? ${numbers.contains(4)}")
  println("Contains 7? ${numbers.contains(7)}")
}
```

7. Uruchom swój kod. To jest wyjście.

```
List: [1, 2, 3, 4, 5, 6]
Size: 6
First element: 1
Second element: 2
Last index: 5
Last element: 6
First: 1
Last: 6
Contains 4? true
Contains 7? false
```

Listy są tylko do odczytu

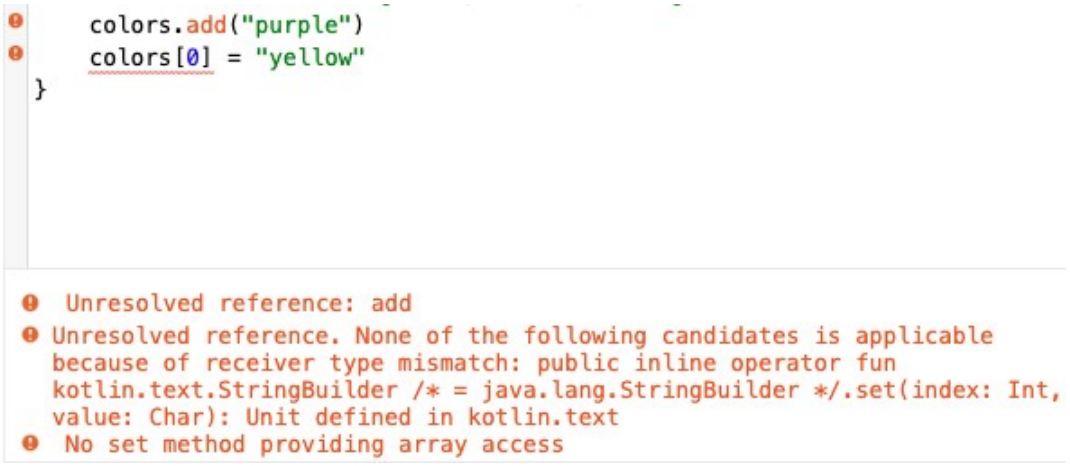
1. Usuń kod w Kotlin Playground i zastąp następującym kodem. Lista `colors`jest inicjowana listą 3 kolorów reprezentowanych jako `Strings`.

```
fun main() {
   val colors = listOf("green", "orange", "blue")
}
```

2. Pamiętaj, że nie możesz dodawać ani zmieniać elementów w trybie tylko do odczytu `List`. Zobacz, co się stanie, jeśli spróbujesz dodać element do listy lub spróbujesz zmodyfikować element listy, ustawiając go na nową wartość.

```
colors.add("purple")
colors[0] = "yellow"
```

3. Uruchom swój kod, a otrzymasz kilka komunikatów o błędach. Zasadniczo błędy mówią, że `add()`metoda nie istnieje dla `List`, i że nie możesz zmienić wartości elementu.



4. Usuń błędny kod.

Widziałeś na własne oczy, że nie można zmienić listy tylko do odczytu. Istnieje jednak szereg operacji na listach, które nie zmieniają listy, ale zwracają nową listę. Dwa z nich to `reversed()`i `sorted()`. Funkcja `reversed()`zwraca nową listę, w której elementy są w odwrotnej kolejności, oraz `sorted()`nową listę, w której elementy są posortowane w porządku rosnącym.

1. Dodaj kod, aby odwrócić `colors`listę. Wydrukuj dane wyjściowe. To jest nowa lista zawierająca elementy `colors`w odwrotnej kolejności.
2. Dodaj drugi wiersz kodu, aby wydrukować oryginał `list`, aby zobaczyć, że oryginalna lista nie uległa zmianie.

```
println("Reversed list: ${colors.reversed()}")
println("List: $colors")
```

3. To jest wynik dwóch wydrukowanych list.

```
Reversed list: [blue, orange, green]
List: [green, orange, blue]
```

4. Dodaj kod, aby zwrócić posortowaną wersję za `List`pomocą `sorted()`funkcji.

```
println("Sorted list: ${colors.sorted()}")
```

Wynikiem jest nowa lista kolorów posortowana w kolejności alfabetycznej. Chłodny!

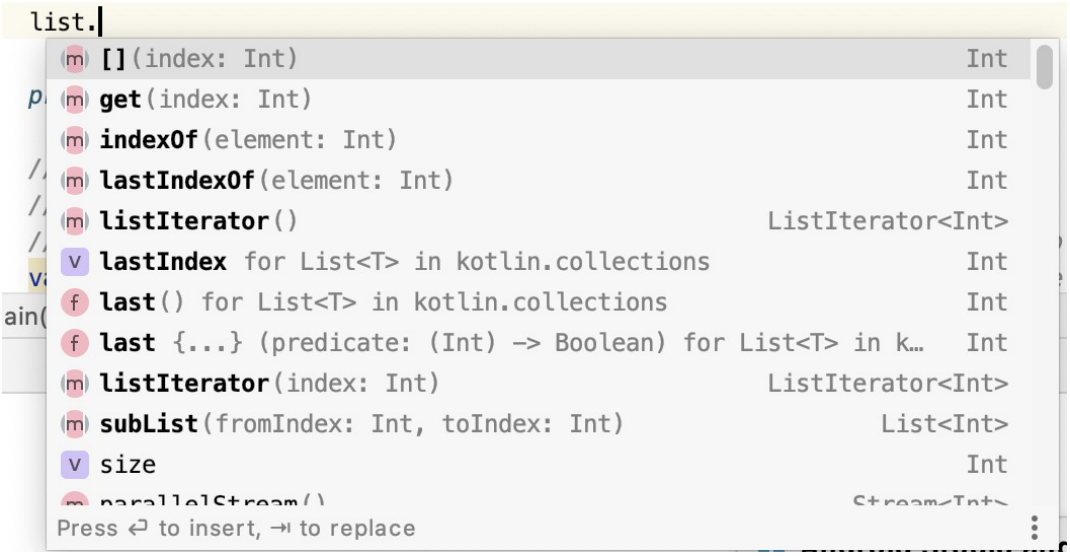
```
Sorted list: [blue, green, orange]
```

5. Możesz także wypróbować `sorted()`funkcję na liście nieposortowanych numerów.

```
val oddNumbers = listOf(5, 3, 7, 1)
println("List: $oddNumbers")
println("Sorted list: ${oddNumbers.sorted()}")
```

```
List: [5, 3, 7, 1]
Sorted list: [1, 3, 5, 7]
```

Uwaga: Nie martw się, że musisz pamiętać wszystkie możliwe operacje na listach. Kiedy tworzysz aplikacje w Android Studio, gdy pracujesz z listami i innymi typami danych, Android Studio pokaże Ci dostępne dla nich funkcje i właściwości, jak na przykład na poniższym zrzucie ekranu.



Do tej pory widać przydatność możliwości tworzenia list. Jednak fajnie byłoby móc modyfikować listę po utworzeniu, więc spójrzmy dalej na listy mutowalne.

3. Wprowadzenie do list mutowalnych

Listy mutowalne to listy, które można modyfikować po utworzeniu. Możesz dodawać, usuwać lub zmieniać elementy. Możesz zrobić wszystko, co możesz zrobić z listami tylko do odczytu. Listy mutowalne są typu MutableListi można je tworzyć przez wywołanie mutableListOf().

Utwórz listę mutowalną

1. Usuń istniejący kod w `main()`.
2. W ramach `main()`funkcji utwórz pustą listę mutowalną i przypisz ją do `val`zmiennej o nazwie `entrees`.

```
val entrees = mutableListOf()
```

Powoduje to następujący błąd, jeśli spróbujesz uruchomić swój kod.

```
Not enough information to infer type variable T
```

Jak wspomniano wcześniej, kiedy tworzysz `MutableList`lub `List`, Kotlin próbuje wywnioskować, jaki typ elementów zawiera lista z przekazanych argumentów. Na przykład, jeśli piszesz `listOf("noodles")`, Kotlin wywnioskuje, że chcesz utworzyć listę `String`. Kiedy inicjujesz pustą listę bez elementów, Kotlin nie może wywnioskować typu elementów, więc musisz wyraźnie określić typ. Zrób to, dodając tekst w nawiasach ostrych zaraz po `mutableListOf`lub `listOf`. (W dokumentacji możesz zobaczyć to jako `<T>`gdzie `T`oznacza parametr typu).

3. Popraw deklarację zmiennej, aby określić, że chcesz utworzyć listę mutowalną typu `String`.

```
val entrees = mutableListOf<String>()
```

Innym sposobem naprawienia błędu jest określenie z góry typu danych zmiennej.

```
val entrees: MutableList<String> = mutableListOf()
```

Uwaga: Możesz użyć `val`dla listy mutowalnej, ponieważ `entrees`zmienna zawiera odniesienie do listy, a to odniesienie nie zmienia się, nawet jeśli zawartość listy zmienia.

4. Wydrukuj listę.

```
println("Entrees: $entrees")
```

5. Dane wyjściowe pokazują `[]`pustą listę.

```
Entrees: []
Dodaj elementy do listy
```

Listy mutowalne stają się interesujące, gdy dodajesz, usuwasz i aktualizujesz elementy.

1. Dodaj `"noodles"`do listy za pomocą `entrees.add("noodles")`.Funkcja `add()`zwraca `true`, jeśli dodanie elementu do listy powiodło się, w `false`przeciwnym razie.
2. Wydrukuj listę, aby potwierdzić, że `"noodles"`rzeczywiście została dodana.

```
println("Add noodles: ${entrees.add("noodles")}")
println("Entrees: $entrees")
```

Dane wyjściowe to:

```
Add noodles: true
Entrees: [noodles]
```

3. Dodaj kolejną pozycję `"spaghetti"`do listy.

```
println("Add spaghetti: ${entrees.add("spaghetti")}")
println("Entrees: $entrees")
```

Otrzymana entreeslista zawiera teraz dwa elementy.

```
Add spaghetti: true
Entrees: [noodles, spaghetti]
```

Zamiast dodawać elementy jeden po drugim za pomocą add(), możesz dodać wiele elementów naraz, używając addAll()i przekazując listę.

4. Utwórz listę moreItems. Nie będziesz musiał tego zmieniać, więc uczyń go valniezmiennym.

```
val moreItems = listOf("ravioli", "lasagna", "fettuccine")
```

5. Używając addAll(), dodaj wszystkie pozycje z nowej listy do entrees. Wydrukuj listę wynikową.

```
println("Add list: ${entrees.addAll(moreItems)}")
println("Entrees: $entrees")
```

Dane wyjściowe pokazują, że dodanie listy powiodło się. Lista entreeszawiera teraz łącznie 5 pozycji.

```
Add list: true
Entrees: [noodles, spaghetti, ravioli, lasagna, fettuccine]
```

6. Teraz spróbuj dodać numer do tej listy.

```
entrees.add(10)
```

To kończy się błędem:

```
The integer literal does not conform to the expected type String
```

Dzieje się tak, ponieważ entreeslista oczekuje elementów typu Stringi próbujesz dodać Int. Pamiętaj, aby do listy dodawać tylko elementy o poprawnym typie danych. W przeciwnym razie otrzymasz błąd kompilacji. Jest to jeden ze sposobów, w jaki Kotlin zapewnia, że Twój kod jest bezpieczniejszy dzięki bezpieczeństwu typów.

7. Usuń niepoprawny wiersz kodu, aby Twój kod się skompilował.

Usuń elementy z listy

1. Zadzwoń remove(), aby usunąć "spaghetti"z listy. Wydrukuj listę ponownie.

```
println("Remove spaghetti: ${entrees.remove("spaghetti")}")
println("Entrees: $entrees")
```

2. Usunięcie "spaghetti"zwraca prawdę, ponieważ element był obecny na liście i mógł zostać pomyślnie usunięty. Na liście pozostały już tylko 4 pozycje.

```
Remove spaghetti: true
Entrees: [noodles, ravioli, lasagna, fettuccine]
```

3. Co się stanie, jeśli spróbujesz usunąć element, którego nie ma na liście? Spróbuj usunąć "rice"z listy za pomocą entrees.remove("rice").

```
println("Remove item that doesn't exist: ${entrees.remove("rice")}")
println("Entrees: $entrees")
```

Metoda `remove()`zwraca `false`, ponieważ element nie istnieje i dlatego nie można go usunąć. Lista pozostaje niezmieniona, wciąż pozostają tylko 4 pozycje. Wyjście:

```
Remove item that doesn't exist: false
Entrees: [noodles, ravioli, lasagna, fettuccine]
```

4. Możesz również określić indeks elementu do usunięcia. Służy `removeAt()`do usuwania pozycji z indeksu `0`.

```
println("Remove first element: ${entrees.removeAt(0)}")
println("Entrees: $entrees")
```

Zwracaną wartością `removeAt(0)`jest pierwszy element (`"noodles"`), który został usunięty z listy. Lista `entrees`zawiera teraz 3 pozostałe pozycje.

```
Remove first element: noodles
Entrees: [ravioli, lasagna, fettuccine]
```

5. Jeśli chcesz wyczyścić całą listę, możesz zadzwonić `clear()`.

```
entrees.clear()
println("Entrees: $entrees")
```

Dane wyjściowe pokazują teraz pustą listę.

```
Entrees: []
```

6. Kotlin daje Ci możliwość sprawdzenia, czy lista jest pusta za pomocą `isEmpty()`funkcji. Spróbuj wydrukować`entrees.isEmpty()`.

```
println("Empty? ${entrees.isEmpty()}")
```

Dane wyjściowe powinny być prawdziwe, ponieważ lista jest obecnie pusta i zawiera 0 elementów.

```
Empty? true
```

Metoda `isEmpty()`jest przydatna, jeśli chcesz wykonać operację na liście lub uzyskać dostęp do określonego elementu, ale najpierw chcesz się upewnić, że lista nie jest pusta.

Oto cały kod, który napisałeś dla mutowalnych list. Komentarze są opcjonalne.

```
fun main() {
  val entrees = mutableListOf<String>()
  println("Entrees: $entrees")

  // Add individual items using add()
  println("Add noodles: ${entrees.add("noodles")}")
  println("Entrees: $entrees")
  println("Add spaghetti: ${entrees.add("spaghetti")}")
  println("Entrees: $entrees")

  // Add a list of items using addAll()
  val moreItems = listOf("ravioli", "lasagna", "fettuccine")
  println("Add list: ${entrees.addAll(moreItems)}")
  println("Entrees: $entrees")
```

```
    // Remove an item using remove()
    println("Remove spaghetti: ${entrees.remove("spaghetti")}")
    println("Entrees: $entrees")
    println("Remove item that doesn't exist: ${entrees.remove("rice")}")
    println("Entrees: $entrees")

    // Remove an item using removeAt() with an index
    println("Remove first element: ${entrees.removeAt(0)}")
    println("Entrees: $entrees")

    // Clear out the list
    entrees.clear()
    println("Entrees: $entrees")

    // Check if the list is empty
    println("Empty? ${entrees.isEmpty()}")
}
```

4. Zapętl listę

Aby wykonać operację na każdym elemencie na liście, możesz przejść przez listę w pętli (znane również jako iteracja po liście). Pętle mogą być używane z `Lists` i `MutableLists`.

Podczas gdy pętle

Jednym z rodzajów pętli jest `while` pętla. Pętla `while` zaczyna się od `while` słowa kluczowego w Kotlinie. Zawiera blok kodu (w nawiasach klamrowych), który jest wykonywany w kółko, o ile wyrażenie w nawiasach jest prawdziwe. Aby zapobiec wykonywaniu kodu w nieskończoność (tzw. pętla nieskończona), blok kodu musi zawierać logikę, która zmienia wartość wyrażenia, dzięki czemu ostatecznie wyrażenie będzie fałszywe i przestaniesz wykonywać pętlę. W tym momencie wychodzisz z `while` pętli i kontynuujesz wykonywanie kodu, który pojawia się po pętli.

```
while (expression) {
    // While the expression is true, execute this code block
}
```

Uwaga: Pętla `while` nie musi obejmować listy (przykład tutaj), ale jest przydatna w przypadku list.

Użyj `while` pętli, aby przejść przez listę. Utwórz zmienną, aby śledzić to, na co `index` aktualnie patrzysz na liście. Ta `index` zmienna będzie zwiększać się o 1 za każdym razem, aż dojdziesz do ostatniego indeksu listy, po czym wyjdziesz z pętli.

1. Usuń istniejący kod w Kotlin Playground i miej pustą `main()` funkcję.
2. Załóżmy, że organizujesz przyjęcie. Utwórz listę, w której każdy element reprezentuje liczbę gości, którzy otrzymali RSVP z każdej rodziny. Pierwsza rodzina powiedziała, że 2 osoby będą pochodzić z ich rodziny. Druga rodzina powiedziała, że przyjedzie 4 z nich i tak dalej.

```
val guestsPerFamily = listOf(2, 4, 1, 3)
```

3. Dowiedz się, ilu będzie łącznie gości. Napisz pętlę, aby znaleźć odpowiedź. Utwórz a `var`dla całkowitej liczby gości i zainicjuj go na `0`.

   ```
   var totalGuests = 0
   ```

4. Zainicjuj a `var`dla `index`zmiennej, jak opisano wcześniej.

   ```
   var index = 0
   ```

5. Napisz `while`pętlę, aby przejść przez listę. Warunkiem jest kontynuowanie wykonywania bloku kodu, dopóki `index`wartość jest mniejsza niż rozmiar listy.

   ```
   while (index < guestsPerFamily.size) {

   }
   ```

6. Wewnątrz pętli pobierz element listy w bieżącym stanie `index`i dodaj go do zmiennej całkowitej liczby gości. Pamiętaj, że `totalGuests += guestsPerFamily[index]`to to samo, co`totalGuests = totalGuests + guestsPerFamily[index]`.

   Zwróć uwagę, że ostatni wiersz pętli zwiększa wartość `index`zmiennej o 1 przy użyciu `index++`, tak że następna iteracja pętli spowoduje wyświetlenie kolejnej rodziny na liście.

   ```
   while (index < guestsPerFamily.size) {
     totalGuests += guestsPerFamily[index]
     index++
   }
   ```

7. Po `while`pętli możesz wydrukować wynik.

   ```
   while ... {
     ...
   }
   println("Total Guest Count: $totalGuests")
   ```

8. Uruchom program, a dane wyjściowe są następujące. Możesz sprawdzić, czy to poprawna odpowiedź, ręcznie dodając liczby na liście.

   ```
   Total Guest Count: 10
   ```

   Oto pełny fragment kodu:

   ```
   val guestsPerFamily = listOf(2, 4, 1, 3)
   var totalGuests = 0
   var index = 0
   while (index < guestsPerFamily.size) {
     totalGuests += guestsPerFamily[index]
     index++
   ```

```
}
println("Total Guest Count: $totalGuests")
```

Uwaga: Ponieważ wykonywanie prostej operacji na zmiennej i przechowywanie jej z powrotem w zmiennej jest powszechne, istnieje skrót do tego przy użyciu `+=`operatora. Istnieją równoważne operatory dla minus (`-=`), mnożenia (`*=`) i dzielenia (`/=`). Zobacz więcej szczegółów tutaj .

Za pomocą `while`pętli trzeba było napisać kod, aby utworzyć zmienną do śledzenia indeksu, pobrać element z indeksu na liście i zaktualizować tę zmienną indeksu. Istnieje jeszcze szybszy i bardziej zwięzły sposób przeglądania listy. Użyj `for`pętli!

Do pętli

Pętla `for`to inny rodzaj pętli. To znacznie ułatwia przeglądanie listy. Zaczyna się od `for`słowa kluczowego w Kotlin z blokiem kodu w nawiasach klamrowych. Warunek wykonania bloku kodu jest podany w nawiasach.

```
for (number in numberList) {
  // For each element in the list, execute this code block
}
```

W tym przykładzie zmienna `number`jest ustawiona jako równa pierwszemu elementowi `numberList`i blok kodu jest wykonywany. Następnie `number`zmienna jest automatycznie aktualizowana, aby była kolejnym elementem `numberList`, a blok kodu jest wykonywany ponownie. Powtarza się to dla każdego elementu listy, aż do osiągnięcia końca `numberList`.

1. Usuń istniejący kod w Kotlin Playground i zastąp następującym kodem:

```
fun main() {
  val names = listOf("Jessica", "Henry", "Alicia", "Jose")
}
```

2. Dodaj `for`pętlę, aby wydrukować wszystkie elementy na `names`liście.

```
for (name in names) {
  println(name)
}
```

Jest to o wiele prostsze, niż gdybyś musiał napisać to jako `while`pętlę!

3. Dane wyjściowe to:

```
Jessica
Henry
Alicia
Jose
```

Typową operacją na listach jest zrobienie czegoś z każdym elementem listy.

4. Zmodyfikuj pętlę, aby wydrukować również liczbę znaków w imieniu tej osoby. Wskazówka: możesz użyć `length`właściwości a, `String`aby znaleźć liczbę znaków w tym `String`.

```
val names = listOf("Jessica", "Henry", "Alicia", "Jose")
for (name in names) {
   println("$name - Number of characters: ${name.length}")
}
```

Wyjście:

```
Jessica - Number of characters: 7
Henry - Number of characters: 5
Alicia - Number of characters: 6
Jose - Number of characters: 4
```

Kod w pętli nie zmienił oryginału `List`. Wpłynęło to tylko na to, co zostało wydrukowane.

To całkiem fajne, jak możesz napisać instrukcje dotyczące tego, co powinno się stać dla 1 elementu listy, a kod zostanie wykonany dla każdego elementu listy! Korzystanie z pętli może oszczędzić ci wpisywania wielu powtarzających się kodów.

Uwaga: Oto kilka innych wariantów tego, co można zrobić z `for`pętlami, w tym używanie ich z zakresami z określonymi krokami (zamiast zwiększania ich o 1 za każdym razem).

```
for (item in list) print(item) // Iterate over items in a list

for (item in 'b'..'g') print(item) // Range of characters in an alphabet

for (item in 1..5) print(item) // Range of numbers

for (item in 5 downTo 1) print(item) // Going backward

for (item in 3..6 step 2) print(item) // Prints: 35
```

Więcej informacji można znaleźć w dokumentacji wymienionej na końcu tego ćwiczenia z programowania.

Uwaga: Pętla jest strukturą przepływu sterowania, taką jak wyrażenia `if`/ `else`i `when`poznane wcześniej, ponieważ dyktuje ona określony przepływ dla sposobu, w jaki powinien być wykonywany kod.

Teraz, gdy już eksperymentowałeś z tworzeniem i używaniem zarówno list, jak i list mutowalnych oraz nauczyłeś się o pętlach, nadszedł czas, aby zastosować tę wiedzę w przykładowym przypadku użycia!

5. Złóż to wszystko razem

Jeśli chodzi o zamawianie jedzenia w lokalnej restauracji, zwykle w jednym zamówieniu dla klienta jest wiele produktów. Korzystanie z list jest idealne do przechowywania informacji o zamówieniu. Skorzystasz również ze swojej wiedzy o klasach i dziedziczeniu, aby stworzyć bardziej niezawodny i skalowalny program Kotlin, zamiast umieszczać cały kod w `main()`funkcji.

Do kolejnej serii zadań stwórz program Kotlin, który pozwala zamawiać różne kombinacje potraw.

Najpierw spójrz na ten przykładowy wynik końcowego kodu. Czy możesz przeprowadzić burzę mózgów, jakiego rodzaju klasy musiałbyś utworzyć, aby uporządkować wszystkie te dane?

```
Order #1
Noodles: $10
Total: $10

Order #2
Noodles: $10
Vegetables Chef's Choice: $5
Total: $15
```

Z danych wyjściowych zauważysz, że:

- jest lista zamówień
- każde zamówienie ma numer
- każde zamówienie może zawierać listę produktów takich jak makaron i warzywa
- każdy przedmiot ma swoją cenę
- każde zamówienie ma cenę całkowitą, która jest sumą cen poszczególnych pozycji

Możesz utworzyć klasę reprezentującą `Order`, i klasę reprezentującą każdy produkt żywnościowy, taki jak `Noodles`lub `Vegetables`. Możesz to dalej zaobserwować `Noodles`i `Vegetables`mieć pewne podobieństwa, ponieważ oba są produktami spożywczymi i każdy ma swoją cenę. Możesz rozważyć utworzenie `Item`klasy ze wspólnymi właściwościami, z których zarówno `Noodle`klasa, jak i `Vegetable`klasa mogą dziedziczyć. W ten sposób nie musisz powielać logiki zarówno w `Noodle`klasie, jak i `Vegetable`klasie.

1. Otrzymasz następujący kod startowy. Profesjonalni programiści często muszą czytać kod innych osób, na przykład, jeśli dołączają do nowego projektu lub dodają funkcję, którą stworzył ktoś inny. Umiejętność czytania i rozumienia kodu jest ważną umiejętnością.

   Poświęć trochę czasu na przejrzenie tego kodu i zrozumienie, co się dzieje. Skopiuj i wklej ten kod do Kotlin Playground i uruchom go. Pamiętaj, aby usunąć istniejący kod w Kotlin Playground przed wklejeniem nowego kodu. Obserwuj wyniki i zobacz, czy to pomoże ci lepiej zrozumieć kod.

```
open class Item(val name: String, val price: Int)

class Noodles : Item("Noodles", 10)

class Vegetables : Item("Vegetables", 5)

fun main() {
  val noodles = Noodles()
  val vegetables = Vegetables()
  println(noodles)
  println(vegetables)
}
```

2. Powinieneś zobaczyć dane wyjściowe podobne do tego:

```
Noodles@5451c3a8
Vegetables@76ed5528
```

Oto bardziej szczegółowe wyjaśnienie kodu. Pierwsza to klasa o nazwie `Item`, w której konstruktor przyjmuje 2 parametry: a `name`dla elementu (jako String) i a `price`(jako liczbę całkowitą). Obie właściwości nie zmieniają się po ich przekazaniu, dlatego są oznaczone jako `val`. Ponieważ `Item`jest klasą nadrzędną i od niej wywodzą się podklasy, jest ona oznaczona `open`słowem kluczowym.

Konstruktor `Noodles`klasy nie przyjmuje żadnych parametrów, ale rozszerza się od `Item`konstruktora nadklasy i wywołuje go, przekazując `"Noodles"`jako nazwę i cenę 10. `Vegetables`Klasa jest podobna, ale wywołuje konstruktor nadklasy z `"Vegetables"`ceną 5.

Funkcja `main()`inicjuje nowe instancje obiektów `Noodles`i `Vegetables`klas oraz wyświetla je na wyjściu.

Zastąp metodę toString()

Podczas drukowania instancji obiektu na wyjściu `toString()`wywoływana jest metoda obiektu. W Kotlinie każda klasa automatycznie dziedziczy `toString()`metodę. Domyślna implementacja tej metody po prostu zwraca typ obiektu z adresem pamięci dla instancji. `toString()`Aby zwrócić coś bardziej znaczącego i przyjaznego dla użytkownika, należy to zmienić niż `Noodles@5451c3a8`i `Vegetables@76ed5528`.

1. Wewnątrz `Noodles`klasy nadpisz `toString()`metodę i zwróć `name`. Pamiętaj, że `Noodles`dziedziczy `name`właściwość po swojej klasie nadrzędnej `Item`.

```
class Noodles : Item("Noodles", 10) {
  override fun toString(): String {
    return name
  }
}
```

2. Powtórz to samo dla `Vegetables`klasy.

```
class Vegetables() : Item("Vegetables", 5) {
  override fun toString(): String {
    return name
  }
}
```

3. Uruchom swój kod. Wyjście wygląda teraz lepiej:

```
Noodles
Vegetables
```

W następnym kroku zmienisz `Vegetables`konstruktor klasy, aby uwzględniał niektóre parametry, i zaktualizujesz `toString()`metodę, aby odzwierciedlić te dodatkowe informacje.

Dostosuj warzywa w kolejności

Aby zupy z makaronem były ciekawsze, możesz uwzględnić w swoich zamówieniach różne warzywa.

1. W `main()`funkcji zamiast inicjowania `Vegetables`instancji bez argumentów wejściowych, przekaż określone rodzaje warzyw, które klient sobie życzy.

```
fun main() {
  ...
  val vegetables = Vegetables("Cabbage", "Sprouts", "Onion")
```

```
    ...
}
```

Jeśli spróbujesz teraz skompilować swój kod, pojawi się błąd, który mówi:

```
Too many arguments for public constructor Vegetables() defined in Vegetables
```

Przekazujesz teraz 3 argumenty typu String do `Vegetables`konstruktora klasy, więc będziesz musiał zmodyfikować `Vegetables`klasę.

2. Zaktualizuj `Vegetables`nagłówek klasy tak, aby zawierał 3 parametry ciągu, jak pokazano w poniższym kodzie:

```
class Vegetables(val topping1: String,
                 val topping2: String,
                 val topping3: String) : Item ("Vegetables", 5) {
```

3. Teraz twój kod kompiluje się ponownie. Jednak to rozwiązanie sprawdza się tylko wtedy, gdy Twoi klienci chcą zawsze zamawiać dokładnie trzy warzywa. Jeśli klient chce zamówić jeden lub pięć warzyw, to nie ma szczęścia.
4. Zamiast używać właściwości dla każdego warzywa, możesz rozwiązać ten problem, akceptując listę warzyw (która może mieć dowolną długość) w konstruktorze `Vegetables`klasy. `List`Powinien zawierać tylko , `Strings`stąd typ parametru wejściowego to `List<String>`.

```
class Vegetables(val toppings: List<String>) : Item("Vegetables", 5) {
```

Nie jest to najbardziej eleganckie rozwiązanie, ponieważ w `main()`, musiałbyś zmienić swój kod, aby utworzyć listę dodatków przed przekazaniem jej do `Vegetables`konstruktora.

```
Vegetables(listOf("Cabbage", "Sprouts", "Onion"))
```

Jest jeszcze lepszy sposób na rozwiązanie tego.

5. W Kotlinie `vararg`modyfikator umożliwia przekazanie zmiennej liczby argumentów tego samego typu do funkcji lub konstruktora. W ten sposób możesz dostarczyć różne warzywa jako pojedyncze ciągi zamiast listy.

   Zmień definicję klasy `Vegetables`na a `vararg` `toppings`of type `String`.

```
class Vegetables(vararg val toppings: String) : Item("Vegetables", 5) {
```

Uwaga: Tylko jeden parametr może być oznaczony jako `vararg`i jest zwykle ostatnim parametrem na liście.

6. Ten kod w `main()`funkcji będzie teraz działał. Możesz utworzyć `Vegetables`instancję, przekazując dowolną liczbę dopełniających Strings.

```
fun main() {
    ...
    val vegetables = Vegetables("Cabbage", "Sprouts", "Onion")
```

```
    ...
}
```

7. Teraz zmodyfikuj `toString()`metodę `Vegetables`klasy tak, aby zwracała a `String`, które również wymienia dodatki w tym formacie: `Vegetables Cabbage, Sprouts, Onion`.

   Zacznij od nazwy elementu (`Vegetables`). Następnie użyj `joinToString()`metody, aby połączyć wszystkie dodatki w jeden ciąg. Dodaj te dwie części razem za pomocą `+`operatora z odstępem pomiędzy.

```
class Vegetables(vararg val toppings: String) : Item("Vegetables", 5) {
    override fun toString(): String {
        return name + " " + toppings.joinToString()
    }
}
```

8. Uruchom swój program, a wynik powinien wyglądać następująco:

```
Noodles
Vegetables Cabbage, Sprouts, Onion
```

   Uwaga: Aby określić inny separator inny niż przecinek, przekaż żądany ciąg separatora jako argument do `joinToString()`metody. Przykład: `joinToString(" ")`oddzielić każdy element spacją.

9. Pisząc programy, musisz wziąć pod uwagę wszystkie możliwe dane wejściowe. Gdy nie ma argumentów wejściowych do `Vegetables`konstruktora, obsłuż `toString()`metodę w bardziej przyjazny dla użytkownika sposób.

   Ponieważ klient chce warzyw, ale nie powiedział, które z nich, jednym z rozwiązań jest zapewnienie im domyślnych warzyw wybranych przez szefa kuchni.

   Zaktualizuj `toString()`metodę, aby była zwracana `Vegetables Chef's Choice`, jeśli nie zostały przekazane żadne dodatki. Skorzystaj z `isEmpty()`metody, o której dowiedziałeś się wcześniej.

```
override fun toString(): String {
    if (toppings.isEmpty()) {
        return "$name Chef's Choice"
    } else {
        return name + " " + toppings.joinToString()
    }
}
```

10. Zaktualizuj `main()`funkcję, aby przetestować obie możliwości tworzenia `Vegetables`instancji bez żadnych argumentów konstruktora iz kilkoma argumentami.

```
fun main() {
    val noodles = Noodles()
    val vegetables = Vegetables("Cabbage", "Sprouts", "Onion")
    val vegetables2 = Vegetables()
    println(noodles)
    println(vegetables)
    println(vegetables2)
```

```
}
```

11. Sprawdź, czy dane wyjściowe są zgodne z oczekiwaniami.

```
Noodles
Vegetables Cabbage, Sprouts, Onion
Vegetables Chef's Choice
Utwórz zamówienie
```

Teraz, gdy masz już trochę artykułów spożywczych, możesz złożyć zamówienie. Hermetyzuj logikę dla zamówienia w `Order`klasie w swoim programie.

1. Zastanów się, jakie właściwości i metody byłyby rozsądne dla tej `Order`klasy. Jeśli to pomoże, oto kilka przykładowych danych wyjściowych z końcowego kodu.

```
Order #1
Noodles: $10
Total: $10

Order #2
Noodles: $10
Vegetables Chef's Choice: $5
Total: $15

Order #3
Noodles: $10
Vegetables Carrots, Beans, Celery: $5
Total: $15

Order #4
Noodles: $10
Vegetables Cabbage, Onion: $5
Total: $15

Order #5
Noodles: $10
Noodles: $10
Vegetables Spinach: $5
Total: $25
```

2. Być może wymyśliłeś następujące rzeczy:

   Zamówienie klasy

   Właściwości: numer zamówienia, lista pozycji

   Metody: dodaj przedmiot, dodaj wiele przedmiotów, wydrukuj podsumowanie zamówienia (w tym cenę)

3. Skupiając się najpierw na właściwościach, jaki powinien być typ danych każdej właściwości? Czy powinny być publiczne czy prywatne w klasie? Czy powinny być przekazywane jako argumenty, czy zdefiniowane w klasie?

4. Jest wiele sposobów na zaimplementowanie tego, ale tutaj jest jedno rozwiązanie. Utwórz, `class Order`który ma `orderNumber`parametr konstruktora liczby całkowitej.

```
class Order(val orderNumber: Int)
```

5. Ponieważ możesz nie znać z góry wszystkich pozycji w zamówieniu, nie wymagaj przekazywania listy pozycji jako argumentu. Zamiast tego można ją zadeklarować jako zmienną klasy najwyższego poziomu i zainicjować ją jako pustą `MutableList`, która może zawierać elementy typu `Item`. Oznacz zmienną `private`, aby tylko ta klasa mogła bezpośrednio modyfikować tę listę elementów. Zabezpieczy to listę przed modyfikacją w nieoczekiwany sposób przez kod spoza tej klasy.

```
class Order(val orderNumber: Int) {
   private val itemList = mutableListOf<Item>()
}
```

6. Śmiało dodaj też metody do definicji klasy. Możesz wybrać rozsądne nazwy dla każdej metody, a logikę implementacji w każdej metodzie możesz na razie pozostawić pustą. Zdecyduj również, jakie argumenty funkcji i zwracane wartości powinny być wymagane.

```
class Order(val orderNumber: Int) {
  private val itemList = mutableListOf<Item>()

  fun addItem(newItem: Item) {
  }

  fun addAll(newItems: List<Item>) {
  }

  fun print() {
  }
}
```

7. Metoda `addItem()`wydaje się najprostsza, więc najpierw zaimplementuj tę funkcję. Pobiera nowy `Item`, a metoda powinna dodać go do `itemList`.

```
fun addItem(newItem: Item) {
   itemList.add(newItem)
}
```

8. Następnie zaimplementuj `addAll()`metodę. Pobiera listę elementów tylko do odczytu. Dodaj wszystkie te pozycje do wewnętrznej listy pozycji.

```
fun addAll(newItems: List<Item>) {
   itemList.addAll(newItems)
}
```

9. Następnie zaimplementuj `print()`metodę, która wypisuje na wyjściu podsumowanie wszystkich pozycji wraz z ich cenami, a także łączną cenę zamówienia.

   Najpierw wydrukuj numer zamówienia. Następnie użyj pętli, aby przejść przez wszystkie pozycje na liście zamówień. Wydrukuj każdy przedmiot i odpowiednią cenę. Zachowaj również sumę dotychczasowych cen

i dodawaj ją w miarę przeglądania listy. Wydrukuj całkowitą cenę na końcu. Spróbuj sam zaimplementować tę logikę. Jeśli potrzebujesz pomocy, sprawdź poniższe rozwiązanie.

Możesz chcieć dołączyć symbol waluty, aby dane wyjściowe były łatwiejsze do odczytania. Oto jeden ze sposobów wdrożenia rozwiązania. Ten kod używa symbolu waluty $, ale możesz go zmodyfikować na symbol lokalnej waluty.

```
fun print() {
  println("Order #${orderNumber}")
  var total = 0
  for (item in itemList) {
    println("${item}: $${item.price}")
    total += item.price
  }
  println("Total: $${total}")
}
```

Dla każdego `item`w `itemList`, wydrukuj `item`(który wywołuje `toString()`wywołanie w `item`), a następnie `price`element. Również przed pętlą zainicjuj `total`zmienną całkowitą na 0. Następnie kontynuuj dodawanie do sumy, dodając cenę bieżącego elementu do `total`.

Utwórz zamówienia

1. Przetestuj swój kod, tworząc `Order`instancje w `main()`funkcji. Najpierw usuń to, co aktualnie masz w swojej `main()`funkcji.
2. Możesz skorzystać z tych przykładowych zamówień lub stworzyć własne. Eksperymentuj z różnymi kombinacjami pozycji w zamówieniach, upewniając się, że przetestowałeś wszystkie ścieżki kodu w swoim kodzie. Na przykład test `addItem()`i `addAll()`metody w `Order`klasie, tworzenie `Vegetables`instancji bez argumentów i z argumentami i tak dalej.

```
fun main() {
  val order1 = Order(1)
  order1.addItem(Noodles())
  order1.print()

  println()

  val order2 = Order(2)
  order2.addItem(Noodles())
  order2.addItem(Vegetables())
  order2.print()

  println()

  val order3 = Order(3)
  val items = listOf(Noodles(), Vegetables("Carrots", "Beans", "Celery"))
  order3.addAll(items)
  order3.print()
```

```
}
```

3. Dane wyjściowe dla powyższego kodu powinny wyglądać następująco. Sprawdź, czy łączna cena jest sumowana prawidłowo.

```
Order #1
Noodles: $10
Total: $10

Order #2
Noodles: $10
Vegetables Chef's Choice: $5
Total: $15

Order #3
Noodles: $10
Vegetables Carrots, Beans, Celery: $5
Total: $15
```

Dobra robota! Wygląda teraz jak zamówienia na jedzenie!

6. Popraw swój kod

Prowadź listę zamówień

Jeśli tworzysz program, który faktycznie będzie używany w sklepie z makaronem, rozsądne byłoby śledzenie listy wszystkich zamówień klientów.

1. Utwórz listę do przechowywania wszystkich zamówień. Czy jest to lista tylko do odczytu, czy lista mutowalna?
2. Dodaj ten kod do `main()`funkcji. Zainicjuj listę tak, aby była na początku pusta. Następnie po utworzeniu każdego zamówienia dodaj zamówienie do listy.

```
fun main() {
  val ordersList = mutableListOf<Order>()

  val order1 = Order(1)
  order1.addItem(Noodles())
  ordersList.add(order1)

  val order2 = Order(2)
  order2.addItem(Noodles())
  order2.addItem(Vegetables())
  ordersList.add(order2)

  val order3 = Order(3)
```

```
    val items = listOf(Noodles(), Vegetables("Carrots", "Beans", "Celery"))
    order3.addAll(items)
    ordersList.add(order3)
}
```

Ponieważ zamówienia są dodawane z czasem, lista powinna być `MutableList`typu `Order`. Następnie użyj `add()`metody dalej, `MutableList`aby dodać każde zamówienie.

3. Gdy masz już listę zamówień, możesz użyć pętli do wydrukowania każdego z nich. Wydrukuj puste wiersze między zamówieniami, aby wyniki były bardziej czytelne.

```
fun main() {
    val ordersList = mutableListOf<Order>()

    ...

    for (order in ordersList) {
        order.print()
        println()
    }
}
```

Usuwa to zduplikowany kod w naszej `main()`funkcji i sprawia, że kod jest łatwiejszy do odczytania! Dane wyjściowe powinny być takie same jak poprzednio.

Implementuj wzorzec konstruktora dla zamówień

Aby Twój kod Kotlin był bardziej zwięzły, możesz użyć wzorca Builder do tworzenia zamówień. Wzorzec Builder to wzorzec projektowy w programowaniu, który umożliwia budowanie złożonego obiektu w podejściu krok po kroku.

1. Zamiast zwracać `Unit`(lub nic) z metod `addItem()`i w klasie, zwróć zmienioną . Kotlin dostarcza słowo kluczowe do odwoływania się do bieżącej instancji obiektu. W ramach metod i zwracasz prąd , zwracając .`addAll()OrderOrderthisaddItem()addAll()Orderthis`

```
fun addItem(newItem: Item): Order {
    itemList.add(newItem)
    return this
}

fun addAll(newItems: List<Item>): Order {
    itemList.addAll(newItems)
    return this
}
```

2. W `main()`funkcji można teraz połączyć wywołania razem, jak pokazano w poniższym kodzie. Ten kod tworzy nowy `Order`i wykorzystuje wzorzec Builder.

```
val order4 = Order(4).addItem(Noodles()).addItem(Vegetables("Cabbage", "Onion"))
ordersList.add(order4)
```

`Order(4)`zwraca `Order`instancję, którą możesz następnie wywołać `addItem(Noodles())`. Metoda `addItem()`zwraca tę samą `Order`instancję (z nowym stanem) i można ją `addItem()`ponownie wywołać z warzywami. Zwrócony `Order`wynik może być przechowywany w `order4`zmiennej.

Istniejący kod do stworzenia `Orders`nadal działa, więc można go pozostawić bez zmian. Chociaż łączenie tych wywołań nie jest obowiązkowe, jest to powszechna i zalecana praktyka, która wykorzystuje wartość zwracaną przez funkcję.

3. W tym momencie nie musisz nawet przechowywać zamówienia w zmiennej. W `main()`funkcji (przed ostatnią pętlą do wydrukowania zamówień) utwórz `Order`bezpośrednio i dodaj do `orderList`. Kod jest również łatwiejszy do odczytania, jeśli każde wywołanie metody zostanie umieszczone w osobnym wierszu.

```
ordersList.add(
  Order(5)
    .addItem(Noodles())
    .addItem(Noodles())
    .addItem(Vegetables("Spinach")))
```

4. Uruchom swój kod, a to jest oczekiwany wynik:

```
Order #1
Noodles: $10
Total: $10

Order #2
Noodles: $10
Vegetables Chef's Choice: $5
Total: $15

Order #3
Noodles: $10
Vegetables Carrots, Beans, Celery: $5
Total: $15

Order #4
Noodles: $10
Vegetables Cabbage, Onion: $5
Total: $15

Order #5
Noodles: $10
Noodles: $10
Vegetables Spinach: $5
Total: $25
```

Gratulujemy ukończenia tego ćwiczenia z programowania!

Teraz przekonałeś się, jak przydatne może być przechowywanie danych na listach, mutowanie list i przeglądanie list. Wykorzystaj tę wiedzę w kontekście aplikacji na Androida, aby wyświetlić listę danych na ekranie podczas następnego ćwiczenia z programowania!

7. Kod rozwiązania

Oto kod rozwiązania dla klas `Item`, `Noodles`, `Vegetables`i . `Order`Funkcja `main()`pokazuje również, jak korzystać z tych klas. Istnieje wiele podejść do implementacji tego programu, więc Twój kod może się nieco różnić.

```
open class Item(val name: String, val price: Int)

class Noodles : Item("Noodles", 10) {
  override fun toString(): String {
    return name
  }
}

class Vegetables(vararg val toppings: String) : Item("Vegetables", 5) {
  override fun toString(): String {
    if (toppings.isEmpty()) {
      return "$name Chef's Choice"
    } else {
      return name + " " + toppings.joinToString()
    }
  }
}

class Order(val orderNumber: Int) {
  private val itemList = mutableListOf<Item>()

  fun addItem(newItem: Item): Order {
    itemList.add(newItem)
    return this
  }

  fun addAll(newItems: List<Item>): Order {
    itemList.addAll(newItems)
    return this
  }
```

```
    fun print() {
        println("Order #${orderNumber}")
        var total = 0
        for (item in itemList) {
            println("${item}: $${item.price}")
            total += item.price
        }
        println("Total: $${total}")
    }
}

fun main() {
    val ordersList = mutableListOf<Order>()

    // Add an item to an order
    val order1 = Order(1)
    order1.addItem(Noodles())
    ordersList.add(order1)

    // Add multiple items individually
    val order2 = Order(2)
    order2.addItem(Noodles())
    order2.addItem(Vegetables())
    ordersList.add(order2)

    // Add a list of items at one time
    val order3 = Order(3)
    val items = listOf(Noodles(), Vegetables("Carrots", "Beans", "Celery"))
    order3.addAll(items)
    ordersList.add(order3)

    // Use builder pattern
    val order4 = Order(4)
        .addItem(Noodles())
        .addItem(Vegetables("Cabbage", "Onion"))
    ordersList.add(order4)

    // Create and add order directly
    ordersList.add(
        Order(5)
            .addItem(Noodles())
            .addItem(Noodles())
            .addItem(Vegetables("Spinach"))
    )
```

```
    // Print out each order
    for (order in ordersList) {
        order.print()
        println()
    }
}
```

8. Podsumowanie

Kotlin zapewnia funkcjonalność, która pomaga w łatwiejszym zarządzaniu kolekcjami danych i manipulowaniu nimi za pośrednictwem Biblioteki Standardowej Kotlin. Kolekcję można zdefiniować jako kilka obiektów tego samego typu danych. W Kotlinie istnieją różne podstawowe typy kolekcji: listy, zestawy i mapy. To ćwiczenie z programowania koncentrowało się w szczególności na listach, a dowiesz się więcej o zestawach i mapach w przyszłych ćwiczeniach z programowania.

- Lista to uporządkowana kolekcja elementów określonego typu, taka jak listaStrings.
- Indeks jest liczbą całkowitą, która odzwierciedla pozycję elementu (np myList[2].).
- Na liście pierwszy element ma indeks 0 (np myList[0].), a ostatni myList.size-1(np . myList[myList.size-1]lub myList.last()).
- Istnieją dwa rodzaje list: ListiMutableList.
- A Listjest tylko do odczytu i nie można go modyfikować po zainicjowaniu. Można jednak zastosować operacje, takie jak sorted()i reversed(), które zwracają nową listę bez zmiany oryginału.
- A MutableListmożna modyfikować po utworzeniu, np. dodając, usuwając lub modyfikując elementy.
- Możesz dodać listę elementów do listy mutowalnej za pomocą addAll().
- Użyj whilepętli, aby wykonać blok kodu, aż wyrażenie zwróci wartość false i zamkniesz pętlę.

  ```
  while (expression) {

  // While the expression is true, execute this code block

  }
  ```

- Użyj forpętli, aby wykonać iterację po wszystkich elementach listy:

  ```
  for (item in myList) {

  // Execute this code block for each element of the list

  }
  ```

- Modyfikator varargumożliwia przekazanie zmiennej liczby argumentów do funkcji lub konstruktora.

9. Dowiedz się więcej

- Przegląd kolekcji Kotlin

- Lista
- Lista określonych operacji
- Lista - Naucz się Kotlina na przykładach
- Pętla - Naucz się Kotlina na przykładach
- listOf()
- mutableListOf()
- whilepętle
- forpętle
- varargzmienna liczba argumentów
- thissłowo kluczowe
- Przypisania rozszerzone (takie jak +=)

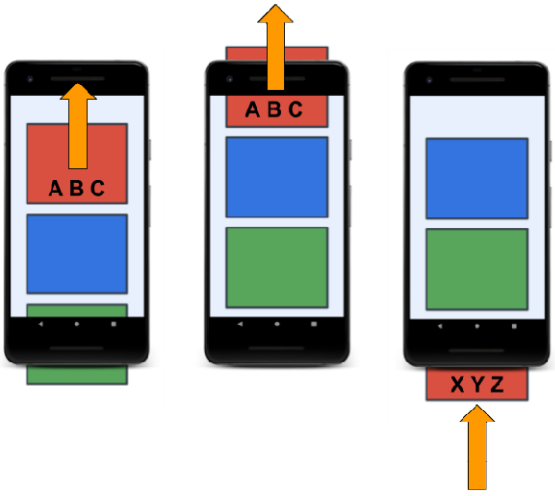
Użyj RecyclerView, aby wyświetlić przewijaną listę

1. Zanim zaczniesz

Jeśli pomyślisz o aplikacjach, których często używasz na telefonie, prawie każda aplikacja ma co najmniej jedną listę. Ekran historii połączeń, aplikacja Kontakty i Twoja ulubiona aplikacja społecznościowa wyświetlają listę danych. Jak pokazano na poniższym zrzucie ekranu, niektóre z tych aplikacji wyświetlają prostą listę słów lub fraz, podczas gdy inne wyświetlają bardziej złożone elementy, takie jak karty zawierające tekst i obrazy. Bez względu na zawartość, wyświetlanie listy danych jest jednym z najczęstszych zadań interfejsu użytkownika w systemie Android.

Aby ułatwić tworzenie aplikacji z listami, system Android udostępnia `RecyclerView`. `RecyclerView`został zaprojektowany tak, aby był bardzo wydajny, nawet w przypadku dużych list, poprzez ponowne wykorzystanie lub recykling widoków, które zostały przewinięte poza ekran. Gdy element listy jest przewijany poza ekranem, `RecyclerView`ponownie wykorzystuje ten widok dla następnego elementu listy, który ma zostać wyświetlony. Oznacza to, że przedmiot jest wypełniony nową zawartością, która przewija się na ekranie. To `RecyclerView`zachowanie oszczędza dużo czasu przetwarzania i pomaga płynniej przewijać listy.

W poniższej kolejności widać, że jeden widok został wypełniony danymi, `ABC`. Po przewinięciu tego widoku z ekranu, `RecyclerView`ponownie wykorzystuje widok dla nowych danych, `XYZ`.



Podczas tego ćwiczenia z programowania zbudujesz aplikację Afirmacje. Afirmacje to prosta aplikacja, która wyświetla dziesięć pozytywnych afirmacji jako tekst na przewijanej liście. Następnie, w kolejnym laboratorium, pójdziesz o krok dalej, dodasz inspirujący obraz do każdej afirmacji i dopracujesz interfejs aplikacji.

Warunki wstępne

- Utwórz projekt z szablonu w Android Studio.
- Dodaj zasoby tekstowe do aplikacji.
- Zdefiniuj układ w XML.
- Rozumienie klas i dziedziczenia w Kotlinie (w tym klas abstrakcyjnych).
- Dziedzicz z istniejącej klasy i zastąp jej metody.
- Skorzystaj z dokumentacji na developer.android.com dla klas dostarczonych przez platformę Android.

Czego się nauczysz

- Jak używać a, `RecyclerView`aby wyświetlić listę danych.
- Jak uporządkować kod w pakiety
- Jak używać adapterów `RecyclerView`do dostosowywania wyglądu poszczególnych elementów listy.

Co zbudujesz

- Aplikacja, która wyświetla listę ciągów afirmacji za pomocą `RecyclerView`.

Czego potrzebujesz

- Komputer z zainstalowanym Android Studio w wersji 4.1 lub nowszej.

2. Tworzenie projektu

Utwórz projekt pustej aktywności

Przed utworzeniem nowego projektu upewnij się, że korzystasz z Android Studio 4.1 lub nowszego.

1. Rozpocznij nowy projekt Kotlin w Android Studio, korzystając z szablonu **Empty Activity .**
2. Wprowadź **Afirmacje jako Nazwę** aplikacji , **com.example.affirmations** jako **Nazwę pakietu** i wybierz **Poziom API 19** jako **Minimalny pakiet SDK** .
3. Kliknij **Zakończ** , aby utworzyć projekt.

3. Konfiguracja listy danych

Następnym krokiem w tworzeniu aplikacji Afirmacje jest dodanie zasobów. Do swojego projektu dodasz następujące elementy.

- Zasoby tekstowe do wyświetlenia jako afirmacje w aplikacji.
- Źródło danych do dostarczenia listy afirmacji do Twojej aplikacji.

Uwaga: w większości projektów produkcyjnych systemu Android można pobrać dane afirmacji z bazy danych lub z serwera. Sieci i bazy danych wykraczają poza zakres tego ćwiczenia z programowania, więc użyjesz listy ciągów afirmacji zdefiniowanych w aplikacji.

Dodaj ciągi afirmacji

1. W oknie **projektu** otwórz **aplikację > res > wartości > strings.xml** . Ten plik ma obecnie jeden zasób, którym jest nazwa aplikacji.
2. W strings.xmlprogramie dodaj następujące afirmacje jako pojedyncze zasoby ciągów. Nazwij je affirmation1, affirmation2i tak dalej.

Tekst afirmacji
I am strong.
I believe in myself.
Each day is a new opportunity to grow and be a better version of myself.
Every challenge in my life is an opportunity to learn from.
I have so much to be grateful for.
Good things are always coming into my life.
New opportunities await me at every turn.
I have the courage to follow my heart.
Things will unfold at precisely the right time.
I will be present in all the moments that this day brings.

Po strings.xmlzakończeniu plik powinien wyglądać tak.

```
<resources>
   <string name="app_name">Affirmations</string>
   <string name="affirmation1">I am strong.</string>
   <string name="affirmation2">I believe in myself.</string>
   <string name="affirmation3">Each day is a new opportunity to grow and be a better version of
myself.</string>
   <string name="affirmation4">Every challenge in my life is an opportunity to learn from.</string>
   <string name="affirmation5">I have so much to be grateful for.</string>
   <string name="affirmation6">Good things are always coming into my life.</string>
   <string name="affirmation7">New opportunities await me at every turn.</string>
   <string name="affirmation8">I have the courage to follow my heart.</string>
   <string name="affirmation9">Things will unfold at precisely the right time.</string>
   <string name="affirmation10">I will be present in all the moments that this day brings.</string>
</resources>
```

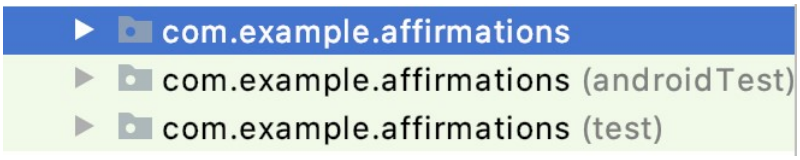
Teraz, gdy dodałeś zasoby ciągów, możesz odwoływać się do nich w swoim kodzie jako R.string.affirmation1lub R.string.affirmation2.

Utwórz nowy pakiet

Logiczne uporządkowanie kodu pomaga Tobie i innym programistom w zrozumieniu, utrzymaniu i rozszerzaniu go. W ten sam sposób, w jaki możesz organizować papierkową robotę w pliki i foldery, możesz organizować swój kod w pliki i pakiety.

Co to jest pakiet?

1. W Android Studio, w oknie **projektu** (**Android**), spójrz na nowe pliki projektu w **aplikacji> java** dla aplikacji Afirmacje. Powinny wyglądać podobnie do poniższego zrzutu ekranu, który pokazuje trzy pakiety, jeden dla twojego kodu (**com.example.affirmations**) i dwa dla plików testowych (**com.example.affirmations (androidTest)** i **com.example.affirmations (test)**) .



2. Zauważ, że nazwa pakietu składa się z kilku słów oddzielonych kropką.

   Istnieją dwa sposoby wykorzystania pakietów.

- Twórz różne pakiety dla różnych części kodu. Na przykład programiści często rozdzielają klasy, które działają z danymi, i klasy, które tworzą interfejs użytkownika, na różne pakiety.
- Użyj kodu z innych pakietów w swoim kodzie. Aby używać klas z innych pakietów, musisz zdefiniować je w zależnościach swojego systemu kompilacji. Jest to również standardowa praktyka `import`w kodzie, więc możesz używać ich skróconych nazw (np. `TextView`) zamiast ich w pełni kwalifikowanych nazw (np `android.widget.TextView`.). Na przykład używałeś już `import`instrukcji dla klas, takich jak `sqrt`(`import kotlin.math.sqrt`) i `View`(`import android.view.View`).

  W aplikacji Afirmacje, oprócz importowania klas Androida i Kotlin, zorganizujesz również swoją aplikację w kilka pakietów. Nawet jeśli nie masz wielu klas dla swojej aplikacji, dobrą praktyką jest używanie pakietów do grupowania klas według funkcjonalności.

Nazewnictwo pakietów

Nazwa pakietu może być dowolna, o ile jest globalnie unikalna; żaden inny opublikowany pakiet nigdzie nie może mieć takiej samej nazwy. Ponieważ istnieje bardzo duża liczba pakietów, a wymyślanie losowych unikalnych nazw jest trudne, programiści stosują konwencje, aby ułatwić tworzenie i zrozumienie nazw pakietów.

- Nazwa pakietu ma zwykle strukturę od ogólnej do szczegółowej, przy czym każda część nazwy jest pisana małymi literami i oddzielona kropką. Ważne: kropka to tylko część nazwy. Nie wskazuje hierarchii w kodzie ani nie nakazuje struktury folderów!
- Ponieważ domeny internetowe są globalnie unikalne, przyjęło się używać domeny, zwykle Twojej lub domeny firmy, jako pierwszej części nazwy.
- Możesz wybrać nazwy pakietów, aby wskazać, co znajduje się w pakiecie i jak pakiety są ze sobą powiązane.
- W przypadku przykładów kodu, takich jak ten, `com.example`często używana jest nazwa aplikacji.

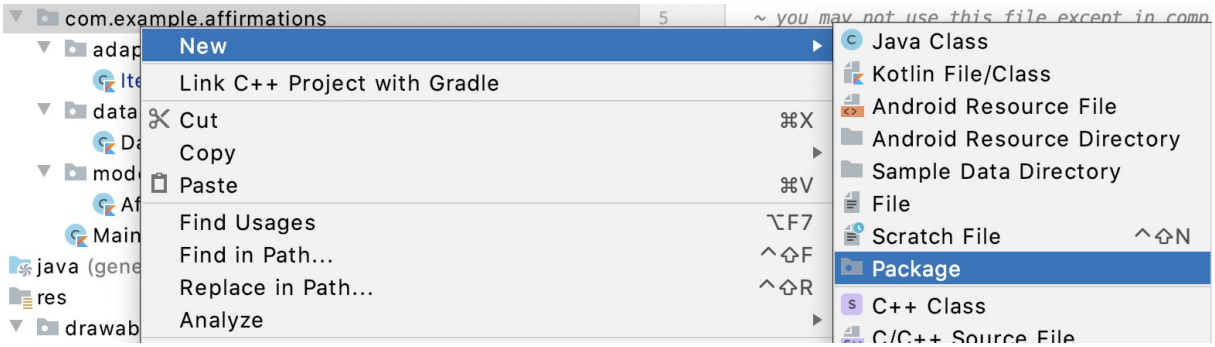
  Oto kilka przykładów predefiniowanych nazw pakietów i ich zawartości:

- `kotlin.math`- Funkcje i stałe matematyczne.
- `android.widget`- Widoki, takie jak `TextView`.

Uwaga: Chociaż nazwy pakietów (i ich rozmieszczenie w oknie Android **Project** w Android Studio jako hierarchia folderów) są wyświetlane jako hierarchia, w kodzie wykonywalnym nie ma rzeczywistej hierarchii. Podobnie jak system numeracji w bibliotece kategoryzuje i porządkuje książki, nadal wszystkie znajdują się na tej samej półce i możesz wyjąć każdą z nich.

Utwórz pakiet

1. W Android Studio w okienku **Projekt** kliknij prawym przyciskiem myszy **app > java > com.example.affirmations** i wybierz **Nowy > Pakiet** .



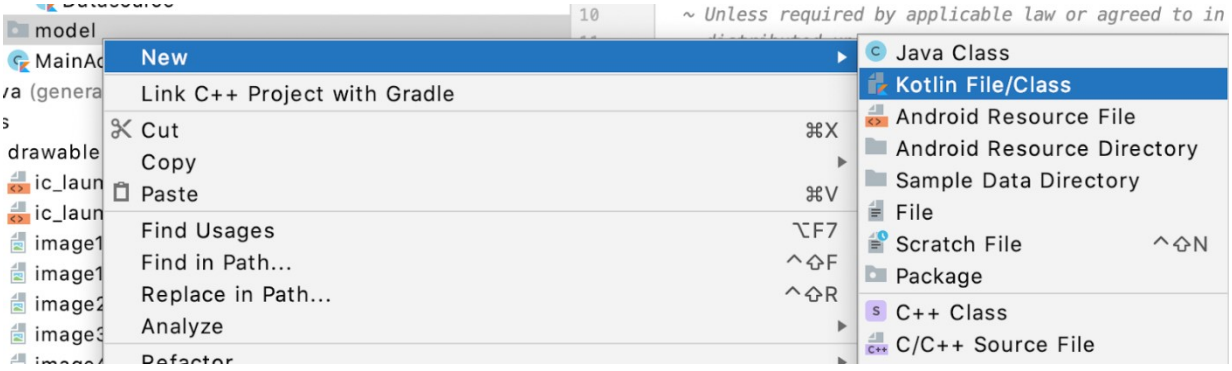
2. W wyskakującym **okienku Nowy pakiet** zwróć uwagę na sugerowany prefiks nazwy pakietu. Sugerowana pierwsza część nazwy pakietu to nazwa pakietu kliknięty prawym przyciskiem myszy. Podczas gdy nazwy pakietów nie tworzą hierarchii pakietów, ponowne użycie części nazwy służy do wskazania relacji i organizacji zawartości!
3. W wyskakującym okienku dołącz **model** na końcu sugerowanej nazwy pakietu. Deweloperzy często używają **modelu** jako nazwy pakietu dla klas, które modelują (lub reprezentują) dane.

4. Naciśnij **Enter** . Spowoduje to utworzenie nowego pakietu w pakiecie **com.example.affirmations** (root). Ten nowy pakiet będzie zawierał wszystkie klasy związane z danymi zdefiniowane w Twojej aplikacji.

Utwórz klasę danych Afirmacja

W tym zadaniu utworzysz klasę o nazwie `Affirmation`.Instancja obiektu `Affirmation`reprezentująca jedną afirmację i zawierającą identyfikator zasobu ciągu z afirmacją.

5. Kliknij prawym przyciskiem myszy pakiet **com.example.affirmations.model i wybierz Nowy> Plik/Klasa Kotlin.**



6. W wyskakującym okienku wybierz **Klasa** i wprowadź `Affirmation`jako nazwę klasy. Tworzy to nowy plik o nazwie `Affirmation.kt`w `model`pakiecie.
7. Utwórz `Affirmation`klasę danych, dodając `data`słowo kluczowe przed definicją klasy. To pozostawia błąd, ponieważ klasy danych muszą mieć zdefiniowaną co najmniej jedną właściwość.

Affirmation.kt

```
package com.example.affirmations.model

data class Affirmation {
}
```

Podczas tworzenia instancji `Affirmation`, musisz przekazać identyfikator zasobu dla ciągu potwierdzenia. Identyfikator zasobu jest liczbą całkowitą.

8. Dodaj `val`parametr całkowity `stringResourceId`do konstruktora `Affirmation`klasy. To pozbywa się twojego błędu.

```
package com.example.affirmations.model

data class Affirmation(val stringResourceId: Int)
```

Utwórz klasę, która będzie źródłem danych

Dane wyświetlane w Twojej aplikacji mogą pochodzić z różnych źródeł (np. w ramach projektu aplikacji lub ze źródła zewnętrznego, które wymaga połączenia z Internetem w celu pobrania danych). W rezultacie dane mogą nie mieć dokładnie takiego formatu, jakiego potrzebujesz. Reszta aplikacji nie powinna zajmować się tym, skąd pochodzą dane ani w jakim formacie są pierwotnie. Możesz i powinieneś ukryć to przygotowanie danych w osobnej `Datasource`klasie, która przygotowuje dane dla aplikacji.

Ponieważ przygotowanie danych to osobna sprawa, umieść `Datasource`klasę w osobnym pakiecie **danych .**

1. W Android Studio, w oknie **Projekt** , kliknij prawym przyciskiem myszy **app > java > com.example.affirmations** i wybierz **Nowy > Pakiet** .
2. Wpisz `data`jako ostatnią część nazwy pakietu.
3. `data`Kliknij pakiet prawym przyciskiem myszy i wybierz **nowy plik/klasę Kotlin.**
4. Wpisz `Datasource`jako nazwę klasy.
5. Wewnątrz `Datasource`klasy utwórz funkcję o nazwie `loadAffirmations()`.

   Funkcja `loadAffirmations()`musi zwrócić listę `Affirmations`. Robisz to, tworząc listę i wypełniając ją `Affirmation`instancją dla każdego ciągu zasobu.

6. Zadeklaruj `List<Affirmation>`jako zwracany typ metody `loadAffirmations()`.
7. W treści `loadAffirmations()`dodaj `return`oświadczenie.
8. Po `return`słowie kluczowym wywołaj `listOf<>()`utworzenie `List`.
9. Wewnątrz nawiasów ostrych `<>`określ typ elementów listy jako `Affirmation`. W razie potrzeby zaimportuj `com.example.affirmations.model.Affirmation`.
10. Wewnątrz nawiasów utwórz `Affirmation`, przekazując `R.string.affirmation1`jako identyfikator zasobu, jak pokazano poniżej.

```
Affirmation(R.string.affirmation1)
```

11. Dodaj pozostałe `Affirmation`obiekty do listy wszystkich afirmacji, oddzielone przecinkami. Gotowy kod powinien wyglądać następująco.

Datasource.kt

```
package com.example.affirmations.data

import com.example.affirmations.R
import com.example.affirmations.model.Affirmation


class Datasource {

  fun loadAffirmations(): List<Affirmation> {
    return listOf<Affirmation>(
      Affirmation(R.string.affirmation1),
      Affirmation(R.string.affirmation2),
      Affirmation(R.string.affirmation3),
      Affirmation(R.string.affirmation4),
      Affirmation(R.string.affirmation5),
      Affirmation(R.string.affirmation6),
      Affirmation(R.string.affirmation7),
      Affirmation(R.string.affirmation8),
      Affirmation(R.string.affirmation9),
      Affirmation(R.string.affirmation10)
    )
  }
}
```

[Opcjonalnie] Wyświetl rozmiar listy afirmacji w TextView

Aby sprawdzić, czy możesz utworzyć listę afirmacji, możesz zadzwonić loadAffirmations()i wyświetlić rozmiar zwróconej listy afirmacji w TextViewszablonie dołączonym do szablonu aplikacji Pusta aktywność.

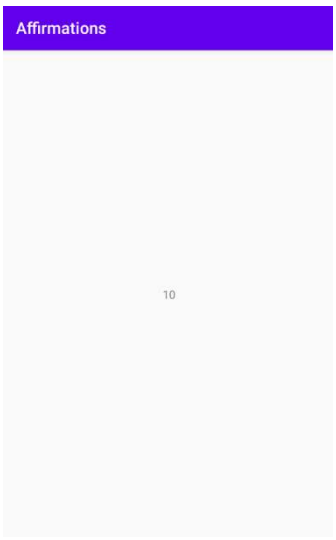
1. W layouts/activity_main.xmlprogramie podaj wartość TextViewpochodzącą z szablonu o idwartości textview.
2. W metodzie MainActivitypo onCreate()istniejącym kodzie pobierz odwołanie do textview.

```
val textView: TextView = findViewById(R.id.textview)
```

3. Następnie dodaj kod, aby utworzyć i wyświetlić rozmiar listy afirmacji. Utwórz Datasource, call loadAffirmations(), pobierz rozmiar zwróconej listy, przekonwertuj ją na ciąg i przypisz jako textof textView.

```
textView.text = Datasource().loadAffirmations().size.toString()
```

4. Uruchom swoją aplikację. Ekran powinien wyglądać tak, jak pokazano poniżej.



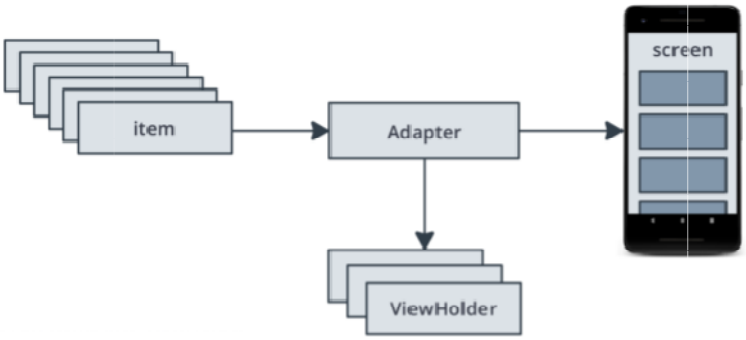
5. Usuń kod dodany przed chwilą w `MainActivity`.

4. Dodawanie RecyclerView do Twojej aplikacji

W tym zadaniu skonfigurujesz `RecyclerView`wyświetlanie listy `Affirmations`.

Istnieje wiele elementów związanych z tworzeniem i używaniem `RecyclerView`. Możesz myśleć o nich jako o podziale pracy. Poniższy diagram przedstawia przegląd, a dowiesz się więcej o każdym z elementów, gdy go zaimplementujesz.

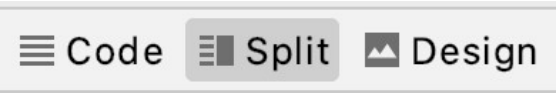
- **element** — jedna pozycja danych z listy do wyświetlenia. Reprezentuje jeden `Affirmation`obiekt w Twojej aplikacji.
- **Adapter** — pobiera dane i przygotowuje je `RecyclerView`do wyświetlenia.
- **ViewHolders** — pula widoków `RecyclerView`do wykorzystania i ponownego wykorzystania do wyświetlania afirmacji.
- **RecyclerView** - Wyświetlenia na ekranie



Dodaj widok RecyclerView do układu

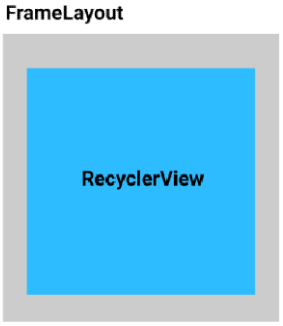
Aplikacja Afirmacje składa się z pojedynczej czynności o nazwie `MainActivity`, a jej plik układu nosi nazwę `activity_main`. `xml`. Najpierw musisz dodać `RecyclerView`do układu `MainActivity`.

1. Otwórz activity_main.xml(**app > res > layout > activity_main.xml**)
2. Jeśli jeszcze go nie używasz, przełącz się na **widok Split** .



3. Usuń TextView.

   Obecny układ wykorzystuje ConstraintLayout. ConstraintLayoutjest idealny i elastyczny, gdy chcesz umieścić wiele widoków podrzędnych w układzie. Ponieważ układ ma tylko jeden widok podrzędny RecyclerView, możesz przełączyć się na prostszy , ViewGroupo nazwie FrameLayout, który powinien być używany do przechowywania jednego widoku podrzędnego.



4. ConstraintLayoutW pliku XML zastąp FrameLayout. Gotowy układ powinien wyglądać tak, jak pokazano poniżej.

   activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout 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=".MainActivity">
</FrameLayout>
```

5. Przełącz na widok **projektu** .
6. W **palecie** wybierz **Kontenery** i znajdź **RecyclerView** .
7. Przeciągnij **RecyclerView** do układu.
8. Jeśli się pojawi, przeczytaj wyskakujące okienko **Dodaj zależność projektu** i kliknij **OK** . (Jeśli wyskakujące okienko się nie pojawi, nie musisz nic robić).
9. Poczekaj, aż Android Studio się zakończy i RecyclerViewpojawi się w układzie.
10. W razie potrzeby zmień atrybuty layout_widthi na tak, aby wypełniały cały ekran.layout_heightRecyclerViewmatch_parentRecyclerView
11. Ustaw identyfikator zasobu RecyclerViewna recycler_view.

RecyclerViewobsługuje wyświetlanie elementów na różne sposoby, takie jak lista liniowa lub siatka. Rozmieszczanie elementów jest obsługiwane przez LayoutManager. Platforma Android zapewnia menedżery układu dla podstawowych układów elementów. Aplikacja Afirmacje wyświetla elementy jako pionową listę, więc możesz użyć LinearLayoutManager.

12. Przełącz się z powrotem do widoku **kodu** . W kodzie XML, wewnątrz RecyclerViewelementu, dodaj LinearLayoutManagerjako atrybut menedżera układu RecyclerView, jak pokazano poniżej.

```
app:layoutManager="LinearLayoutManager"
```

Aby móc przewijać pionową listę elementów, która jest dłuższa niż ekran, musisz dodać pionowy pasek przewijania.

13. Wewnątrz RecyclerViewdodaj android:scrollbarsatrybut ustawiony na vertical.

```
android:scrollbars="vertical"
```

Ostateczny układ XML powinien wyglądać tak:

activity_main.xml

```
<FrameLayout 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=".MainActivity">

  <androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scrollbars="vertical"
    app:layoutManager="LinearLayoutManager" />

</FrameLayout>
```

14. Uruchom swoją aplikację.

Projekt powinien się skompilować i uruchomić bez żadnych problemów. Jednak w aplikacji wyświetlane jest tylko białe tło, ponieważ brakuje kluczowego fragmentu kodu. W tej chwili masz źródło danych i RecyclerViewdodane do układu, ale RecyclerViewnie ma informacji o tym, jak wyświetlić Affirmationobiekty.

Zaimplementuj adapter dla RecyclerView

Twoja aplikacja potrzebuje sposobu, aby pobrać dane z Datasourceprogramu i sformatować je tak, aby każdy z nich Affirmationmógł być wyświetlany jako element w programie RecyclerView.

Adapter to wzorzec projektowy, który dostosowuje dane do czegoś, co może być używane przez program RecyclerView. W takim przypadku potrzebny jest adapter, który pobiera Affirmationinstancję z

listy zwróconej przez `loadAffirmations()`i zamienia ją w widok elementu listy, aby można ją było wyświetlić w`RecyclerView`.

Po uruchomieniu aplikacji `RecyclerView`używa adaptera, aby dowiedzieć się, jak wyświetlić dane na ekranie. `RecyclerView`prosi adapter o utworzenie nowego widoku elementu listy dla pierwszego elementu danych na liście. Po uzyskaniu widoku prosi adapter o podanie danych do narysowania elementu. Ten proces powtarza się do momentu , w którym `RecyclerView`nie trzeba już więcej widoków, aby wypełnić ekran. Jeśli na ekranie zmieszczą się jednocześnie tylko 3 widoki elementów listy, `RecyclerView`adapter prosi tylko o przygotowanie tych 3 widoków elementów listy (zamiast wszystkich 10 widoków elementów listy).

W tym kroku zbudujesz adapter, który dostosuje `Affirmation`instancję obiektu tak, aby można go było wyświetlić w `RecyclerView`.

Utwórz adapter

Adapter składa się z wielu części i napiszesz sporo kodu, który jest bardziej złożony niż to, co do tej pory robiłeś w tym kursie. W porządku, jeśli na początku nie rozumiesz w pełni szczegółów. Po ukończeniu całej aplikacji za pomocą `RecyclerView`, będziesz w stanie lepiej zrozumieć, jak wszystkie części do siebie pasują. Będziesz także mógł ponownie wykorzystać ten kod jako bazę dla przyszłych aplikacji tworzonych za pomocą `RecyclerView`.

Utwórz układ dla przedmiotów

Każdy element w `RecyclerView`ma swój własny układ, który definiujesz w osobnym pliku układu. Ponieważ masz zamiar wyświetlić tylko ciąg znaków, możesz użyć a `TextView`do układu elementu.

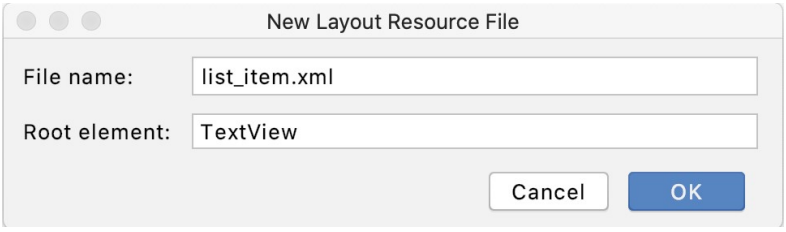
1. W **układzie res >** utwórz nowy pusty **plik** o nazwie `list_item.xml`.
2. Otwórz `list_item.xml`w widoku **Kod** .
3. Dodaj `TextView`z `id` `item_title`.
4. Dodaj `wrap_content`dla `layout_width`i `layout_height`, jak pokazano w poniższym kodzie.

Zauważ, że nie potrzebujesz `ViewGroup`wokół układu, ponieważ ten układ elementu listy zostanie później rozdmuchany i dodany jako element podrzędny do elementu nadrzędnego `RecyclerView`.

`list_item.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/item_title"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content" />
```

Alternatywnie można było użyć **File > New > Layout Resource File** , z **nazwą pliku**`list_item.xml` i `TextView`jako **elementem** głównym . Następnie zaktualizuj wygenerowany kod, aby pasował do powyższego kodu.



Utwórz klasę ItemAdapter

1. W Android Studio w okienku **Projekt** kliknij prawym przyciskiem myszy **app > java > com.example.affirmations** i wybierz **Nowy > Pakiet** .
2. Wpisz adapterjako ostatnią część nazwy pakietu.
3. adapterKliknij pakiet prawym przyciskiem myszy i wybierz **Nowy > Plik/Klasa Kotlin.**
4. Wpisz ItemAdapterjako nazwę klasy, zakończ, a ItemAdapter.ktplik się otworzy.

   Musisz dodać parametr do konstruktora ItemAdapter, aby można było przekazać listę afirmacji do adaptera.

5. Dodaj parametr do ItemAdapterkonstruktora, który jest valwywoływany datasettypu List<Affirmation>. AffirmationW razie potrzeby zaimportuj .
6. Ponieważ the datasetbędzie używany tylko w tej klasie, zrób to private.

   ItemAdapter.kt

```
import com.example.affirmations.model.Affirmation

class ItemAdapter(private val dataset: List<Affirmation>) {

}
```

   Potrzebne ItemAdapterinformacje o tym, jak rozwiązać zasoby tekstowe. Te i inne informacje o aplikacji są przechowywane w Contextinstancji obiektu, którą możesz przekazać do ItemAdapterinstancji.

7. Dodaj parametr do ItemAdapterkonstruktora, który jest valwywoływany contexttypu Context. Ustaw go jako pierwszy parametr w konstruktorze.

```
class ItemAdapter(private val context: Context, private val dataset: List<Affirmation>) {

}
```

Utwórz ViewHolder

RecyclerViewnie wchodzi w bezpośrednią interakcję z widokami przedmiotów, ale zajmuje się ViewHoldersnimi. A ViewHolderreprezentuje widok pojedynczego elementu listy w programie RecyclerViewi może być ponownie użyty, gdy jest to możliwe. Instancja ViewHolderprzechowuje odniesienia do poszczególnych widoków w układzie elementu listy (stąd nazwa „posiadacz widoku"). Ułatwia to aktualizowanie widoku elementu listy o nowe dane. Uchwyty widoków dodają również informacje, które RecyclerViewsłużą do efektywnego przenoszenia widoków na ekranie.

1. Wewnątrz ItemAdapterklasy, przed zamykającym nawiasem klamrowym dla ItemAdapter, utwórz ItemViewHolderklasę.

```
class ItemAdapter(private val context: Context, private val dataset: List<Affirmation>) {

    class ItemViewHolder()
}
```

- Definiowanie klasy wewnątrz innej klasy nazywa się tworzeniem **klasy zagnieżdżonej** .
- Ponieważ `ItemViewHolder`jest używany tylko przez `ItemAdapter`, utworzenie go w środku `ItemAdapter`pokazuje tę zależność. Nie jest to obowiązkowe, ale pomaga innym programistom zrozumieć strukturę Twojego programu.

2. Dodaj `private val viewtyp View`jako parametr do `ItemViewHolder`konstruktora klasy.
3. Utwórz `ItemViewHolder`podklasę `RecyclerView.ViewHolder`i przekaż `view`parametr do konstruktora nadklasy.
4. Wewnątrz `ItemViewHolder`zdefiniuj `val`właściwość `textView`, która jest typu `TextView`. Przypisz mu widok o identyfikatorze `item_title`zdefiniowanym w `list_item. xml`.

```
class ItemAdapter(private val context: Context, private val dataset: List<Affirmation>) {

    class ItemViewHolder(private val view: View) : RecyclerView.ViewHolder(view) {
        val textView: TextView = view.findViewById(R.id.item_title)
    }
}
```

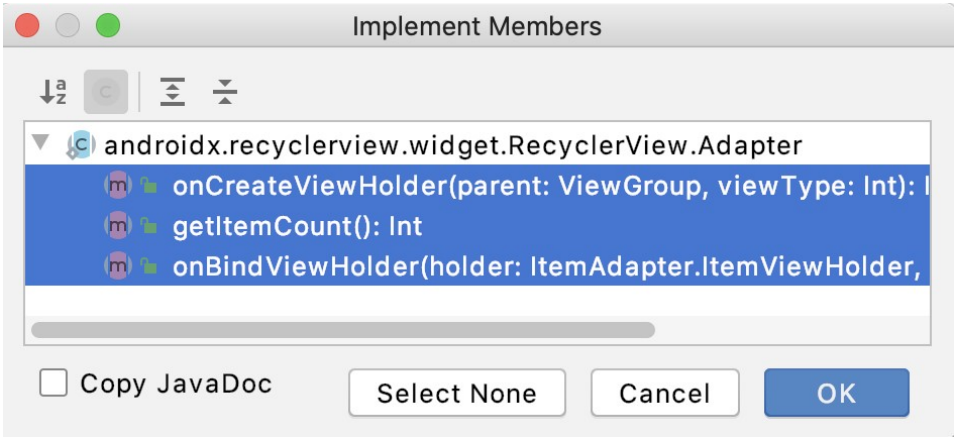
Zastąp metody adaptera

1. Dodaj kod, aby rozszerzyć swoją `ItemAdapter` klasę abstrakcyjną `RecyclerView.Adapter`. Określ `ItemAdapter.ItemViewHolder`jako typ uchwytu widoku w nawiasach ostrych.

```
class ItemAdapter(
    private val context: Context,
    private val dataset: List<Affirmation>
) : RecyclerView.Adapter<ItemAdapter.ItemViewHolder>() {

    class ItemViewHolder(private val view: View) : RecyclerView.ViewHolder(view) {
        val textView: TextView = view.findViewById(R.id.item_title)
    }
}
```

Zobaczysz błąd, ponieważ musisz zaimplementować kilka abstrakcyjnych metod z `RecyclerView.Adapter`.

2. Umieść kursor `ItemAdapter`i naciśnij **Command + I** (**Control + I** w systemie Windows). To pokazuje listę metod, które musisz zaimplementować: `getItemCount()`, `onCreateViewHolder()`i `onBindViewHolder()`.



3. Wybierz wszystkie trzy funkcje za pomocą **Shift+kliknięcie** i kliknij **OK** .

   Tworzy to odgałęzienia z poprawnymi parametrami dla trzech metod, jak pokazano poniżej.

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
  TODO("Not yet implemented")
}

override fun getItemCount(): Int {
  TODO("Not yet implemented")
}

override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
  TODO("Not yet implemented")
}
```

   Nie powinieneś widzieć więcej błędów. Następnie musisz zaimplementować te metody, aby robiły właściwe rzeczy dla Twojej aplikacji.

   Implementuj getItemCount()

   Metoda `getItemCount()`musi zwrócić rozmiar zestawu danych. Dane Twojej aplikacji znajdują się we `dataset`właściwości, którą przekazujesz do `ItemAdapter`konstruktora, i możesz uzyskać ich rozmiar za pomocą `size`.

1. Zastąp `getItemCount()`tym:

```
override fun getItemCount() = dataset.size
```

   Jest to bardziej zwięzły sposób napisania tego:

```
override fun getItemCount(): Int {
  return dataset.size
}
```

   Implementuj onCreateViewHolder()

Metoda onCreateViewHolder()jest wywoływana przez menedżera układu w celu utworzenia nowych uchwytów widoków dla RecyclerView(gdy nie ma istniejących uchwytów widoków, które można ponownie wykorzystać). Pamiętaj, że posiadacz widoku reprezentuje pojedynczy widok elementu listy.

Metoda onCreateViewHolder()przyjmuje dwa parametry i zwraca nowy ViewHolder.

- Parametr parentbędący grupą widoków, do której nowy widok elementu listy zostanie dołączony jako element podrzędny. Rodzicem jest RecyclerView.
- Parametr viewType, który staje się ważny, gdy istnieje wiele typów widoków elementów w tym samym RecyclerView. Jeśli masz różne układy elementów listy wyświetlane w RecyclerView, istnieją różne typy widoków elementów. Możesz przetwarzać tylko widoki z tym samym typem widoku elementu. W Twoim przypadku istnieje tylko jeden układ elementu listy i jeden typ widoku elementu, więc nie musisz się martwić o ten parametr.

1. W onCreateViewHolder()metodzie uzyskaj wystąpienie LayoutInflaterz podanego kontekstu (contextz parent). Inflater układu wie, jak nadmuchać układ XML do hierarchii obiektów widoku.

   ```
   val adapterLayout = LayoutInflater.from(parent.context)
   ```

2. Po utworzeniu LayoutInflaterwystąpienia obiektu dodaj kropkę, po której następuje inne wywołanie metody, aby zwiększyć rzeczywisty widok elementu listy. Przekaż identyfikator zasobu układu XML R.layout.list_itemi parentgrupę widoków. Trzecim argumentem logicznym jest attachToRoot. Ten argument musi być false, ponieważ RecyclerViewdodaje ten element do hierarchii widoków, gdy nadejdzie czas.

   ```
   val adapterLayout = LayoutInflater.from(parent.context)
       .inflate(R.layout.list_item, parent, false)
   ```

   Teraz adapterLayoutzawiera odniesienie do widoku elementu listy (z którego możemy później znaleźć

   widoki podrzędne, takie jak TextView).

3. W programie onCreateViewHolder(), zwróć nową ItemViewHolderinstancję, w której znajduje się widok główny adapterLayout.

   ```
   return ItemViewHolder(adapterLayout)
   ```

   Oto kod do onCreateViewHolder()tej pory.

   ```
   override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
     // create a new view
     val adapterLayout = LayoutInflater.from(parent.context)
       .inflate(R.layout.list_item, parent, false)

     return ItemViewHolder(adapterLayout)
   }
   ```

   Implementuj onBindViewHolder()

   Ostatnią metodą, którą musisz zastąpić, jest onBindViewHolder(). Ta metoda jest wywoływana przez menedżera układu w celu zastąpienia zawartości widoku elementu listy.

Metoda onBindViewHolder()ma dwa parametry, ItemViewHolderwcześniej utworzony przez onCreateViewHolder()metodę i intreprezentujący bieżący element positionna liście. W tej metodzie znajdziesz odpowiedni Affirmationobiekt ze zbioru danych na podstawie pozycji.

1. Wewnątrz onBindViewHolder()stwórz val itemi zdobądź przedmiot na podany positionw dataset.

```
val item = dataset[position]
```

Na koniec musisz zaktualizować wszystkie widoki, do których odwołuje się posiadacz widoku, aby odzwierciedlić prawidłowe dane dla tego elementu. W tym przypadku jest tylko jeden widok: TextViewwnętrze ItemViewHolder. Ustaw tekst, TextViewaby wyświetlić Affirmationciąg dla tego elementu.

2. W przypadku Affirmationwystąpienia obiektu można znaleźć odpowiedni identyfikator zasobu ciągu, wywołując item.stringResourceId. Jest to jednak liczba całkowita i musisz znaleźć mapowanie na rzeczywistą wartość ciągu.

   W systemie Android można wywołać getString()z identyfikatorem zasobu ciągu i zwróci skojarzoną z nim wartość ciągu. getString()jest metodą w Resourcesklasie, a instancję klasy można pobrać za Resourcespomocą context.

   Oznacza to, że możesz wywołać context.resources.getString()i przekazać identyfikator zasobu ciągu. Wynikowy ciąg można ustawić jako textciąg textVieww holder ItemViewHolder. Krótko mówiąc, ten wiersz kodu aktualizuje posiadacza widoku, aby pokazać ciąg afirmacji.

```
holder.textView.text = context.resources.getString(item.stringResourceId)
```

Ukończona onBindViewHolder() metoda powinna wyglądać tak.

```
override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
    val item = dataset[position]
    holder.textView.text =  context.resources.getString(item.stringResourceId)
}
```

Oto gotowy kod adaptera.

ItemAdapter.kt

```
package com.example.affirmations.adapter

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.affirmations.R
import com.example.affirmations.model.Affirmation

/**
```

```
 * Adapter for the [RecyclerView] in [MainActivity]. Displays [Affirmation] data object.
 */
class ItemAdapter(
    private val context: Context,
    private val dataset: List<Affirmation>
) : RecyclerView.Adapter<ItemAdapter.ItemViewHolder>() {

    // Provide a reference to the views for each data item
    // Complex data items may need more than one view per item, and
    // you provide access to all the views for a data item in a view holder.
    // Each data item is just an Affirmation object.
    class ItemViewHolder(private val view: View) : RecyclerView.ViewHolder(view) {
        val textView: TextView = view.findViewById(R.id.item_title)
    }

    /**
     * Create new views (invoked by the layout manager)
     */
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
        // create a new view
        val adapterLayout = LayoutInflater.from(parent.context)
            .inflate(R.layout.list_item, parent, false)

        return ItemViewHolder(adapterLayout)
    }

    /**
     * Replace the contents of a view (invoked by the layout manager)
     */
    override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
        val item = dataset[position]
        holder.textView.text = context.resources.getString(item.stringResourceId)
    }

    /**
     * Return the size of your dataset (invoked by the layout manager)
     */
    override fun getItemCount() = dataset.size
}
```

Teraz, gdy już zaimplementowałeś `ItemAdapter`, musisz powiedzieć użytkownikowi, `RecyclerView`aby używał tego adaptera.

Zmodyfikuj MainActivity, aby użyć RecyclerView

Aby zakończyć, musisz użyć swoich `Datasource`i `ItemAdapter`klas do tworzenia i wyświetlania elementów w `RecyclerView`. Robisz to w `MainActivity`.

1. Otwórz MainActivity.kt.
2. Przejdź MainActivity,do onCreate()metody. Wstaw nowy kod opisany w poniższych krokach po wywołaniusetContentView(R.layout.activity_main).
3. Utwórz wystąpienie Datasourcei wywołaj na loadAffirmations()nim metodę. Przechowuj zwróconą listę afirmacji w valnazwanym myDataset.

   ```
   val myDataset = Datasource().loadAffirmations()
   ```

4. Utwórz zmienną o nazwie recyclerViewi użyj findViewById()jej, aby znaleźć odwołanie do elementu RecyclerVieww układzie.

   ```
   val recyclerView = findViewById<RecyclerView>(R.id.recycler_view)
   ```

5. Aby nakazać recyclerViewużytkownikowi korzystanie z ItemAdapterutworzonej klasy, utwórz nową ItemAdapterinstancję. ItemAdapteroczekuje dwóch parametrów: kontekstu (this) tej czynności oraz afirmacji w myDataset.
6. Przypisz ItemAdapterobiekt do adapterwłaściwości recyclerView.

   ```
   recyclerView.adapter = ItemAdapter(this, myDataset)
   ```

7. Ponieważ rozmiar Twojego układu RecyclerViewjest ustalony w układzie działania, możesz ustawić setHasFixedSizeparametr RecyclerViewna true. To ustawienie jest potrzebne tylko do poprawy wydajności. Użyj tego ustawienia, jeśli wiesz, że zmiany zawartości nie zmieniają rozmiaru układu RecyclerView.

   ```
   recyclerView.setHasFixedSize(true)
   ```

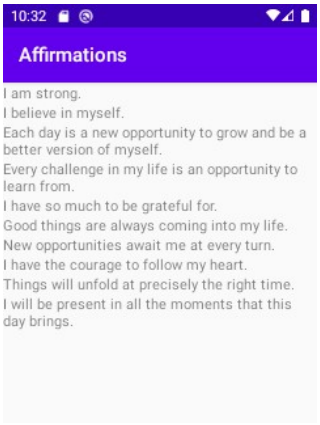
8. Kiedy skończysz, kod MainActivitypowinien być podobny do poniższego.

   MainActivity.kt

   ```
   package com.example.affirmations

   import android.os.Bundle
   import androidx.appcompat.app.AppCompatActivity
   import androidx.recyclerview.widget.RecyclerView
   import com.example.affirmations.adapter.ItemAdapter
   import com.example.affirmations.data.Datasource

   class MainActivity : AppCompatActivity() {

     override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_main)

       // Initialize data.
       val myDataset = Datasource().loadAffirmations()

       val recyclerView = findViewById<RecyclerView>(R.id.recycler_view)
       recyclerView.adapter = ItemAdapter(this, myDataset)
   ```

```
        // Use this setting to improve performance if you know that changes
        // in content do not change the layout size of the RecyclerView
        recyclerView.setHasFixedSize(true)
    }
}
```

9. Uruchom swoją aplikację. Powinieneś zobaczyć listę ciągów afirmacji wyświetlanych na ekranie.



Gratulacje! Właśnie utworzyłeś aplikację wyświetlającą listę danych za `RecyclerView`pomocą niestandardowego adaptera. Poświęć trochę czasu na przejrzenie utworzonego kodu i zrozumienie, jak różne elementy współpracują ze sobą.

Ta aplikacja zawiera wszystkie elementy wymagane do wyświetlenia Twoich afirmacji, ale nie jest jeszcze gotowa do produkcji. Interfejs użytkownika może wymagać pewnych ulepszeń. Podczas następnego ćwiczenia z programowania ulepszysz swój kod, nauczysz się dodawać obrazy do aplikacji i dopracujesz interfejs użytkownika.

5. Kod rozwiązania

Kod rozwiązania dla tego ćwiczenia programowania znajduje się w projekcie i module pokazanym poniżej. Zwróć uwagę, że niektóre pliki Kotlin znajdują się w różnych pakietach, jak wskazuje `package`oświadczenie na początku pliku.

`res/values/strings.xml`

```
<resources>
    <string name="app_name">Affirmations</string>
    <string name="affirmation1">I am strong.</string>
    <string name="affirmation2">I believe in myself.</string>
    <string name="affirmation3">Each day is a new opportunity to grow and be a better version of myself.</string>
    <string name="affirmation4">Every challenge in my life is an opportunity to learn from.</string>
    <string name="affirmation5">I have so much to be grateful for.</string>
    <string name="affirmation6">Good things are always coming into my life.</string>
```

```
    <string name="affirmation7">New opportunities await me at every turn.</string>
    <string name="affirmation8">I have the courage to follow my heart.</string>
    <string name="affirmation9">Things will unfold at precisely the right time.</string>
    <string name="affirmation10">I will be present in all the moments that this day brings.</string>
</resources>
```

affirmations/data/Datasource.kt

```
package com.example.affirmations.data

import com.example.affirmations.R
import com.example.affirmations.model.Affirmation


class Datasource {

    fun loadAffirmations(): List<Affirmation> {
        return listOf<Affirmation>(
            Affirmation(R.string.affirmation1),
            Affirmation(R.string.affirmation2),
            Affirmation(R.string.affirmation3),
            Affirmation(R.string.affirmation4),
            Affirmation(R.string.affirmation5),
            Affirmation(R.string.affirmation6),
            Affirmation(R.string.affirmation7),
            Affirmation(R.string.affirmation8),
            Affirmation(R.string.affirmation9),
            Affirmation(R.string.affirmation10)
        )
    }
}
```

affirmations/model/Affirmation.kt

```
package com.example.affirmations.model

data class Affirmation(val stringResourceId: Int)
```

affirmations/MainActivty.kt

```
package com.example.affirmations

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.RecyclerView
```

```
import com.example.affirmations.adapter.ItemAdapter
import com.example.affirmations.data.Datasource

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Initialize data.
        val myDataset = Datasource().loadAffirmations()

        val recyclerView = findViewById<RecyclerView>(R.id.recycler_view)
        recyclerView.adapter = ItemAdapter(this, myDataset)

        // Use this setting to improve performance if you know that changes
        // in content do not change the layout size of the RecyclerView
        recyclerView.setHasFixedSize(true)
    }
}
```

affirmations/adapter/ItemAdapter.kt

```
package com.example.affirmations.adapter

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.affirmations.R
import com.example.affirmations.model.Affirmation

/**
 * Adapter for the [RecyclerView] in [MainActivity]. Displays [Affirmation] data object.
 */
class ItemAdapter(
    private val context: Context,
    private val dataset: List<Affirmation>
) : RecyclerView.Adapter<ItemAdapter.ItemViewHolder>() {

    // Provide a reference to the views for each data item
    // Complex data items may need more than one view per item, and
    // you provide access to all the views for a data item in a view holder.
    // Each data item is just an Affirmation object.
```

```
    class ItemViewHolder(private val view: View) : RecyclerView.ViewHolder(view) {
        val textView: TextView = view.findViewById(R.id.item_title)
    }

    /**
     * Create new views (invoked by the layout manager)
     */
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
        // create a new view
        val adapterLayout = LayoutInflater.from(parent.context)
            .inflate(R.layout.list_item, parent, false)

        return ItemViewHolder(adapterLayout)
    }

    /**
     * Replace the contents of a view (invoked by the layout manager)
     */
    override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
        val item = dataset[position]
        holder.textView.text = context.resources.getString(item.stringResourceId)
    }

    /**
     * Return the size of your dataset (invoked by the layout manager)
     */
    override fun getItemCount() = dataset.size
}
```

src/main/res/layout/activty_main.xml

```
<FrameLayout 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=".MainActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scrollbars="vertical"
        app:layoutManager="LinearLayoutManager" />
```

```
</FrameLayout>
```

src/main/res/layout/list_item.xml

6. Podsumowanie

- RecyclerViewwidget pomaga wyświetlić listę danych.
- RecyclerViewużywa wzorca adaptera do adaptacji i wyświetlania danych.
- ViewHoldertworzy i przechowuje widoki dla RecyclerView.
- RecyclerViewpochodzi z wbudowanym LayoutManagers. RecyclerViewdeleguje sposób rozmieszczenia elementów w programie LayoutManagers.

Aby zaimplementować adapter:

- Utwórz nową klasę dla adaptera, na przykład ItemAdapter.
- Utwórz ViewHolderklasę niestandardową, która reprezentuje widok pojedynczego elementu listy. Wyjdź z RecyclerView.ViewHolderklasy.
- Zmodyfikuj ItemAdapterklasę, aby rozszerzyła się z RecyclerView. Adapterklasa z klasą niestandardową ViewHolder.
- Zaimplementuj te metody w adapterze: getItemsCount(), onCreateViewHolder()i onBindViewHolder().

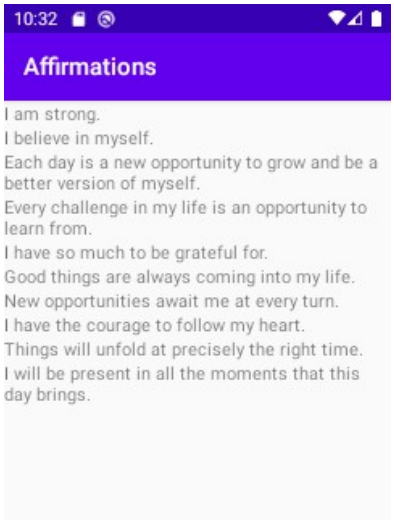
7. Dowiedz się więcej

- Utwórz listę za pomocąRecyclerView
- RecyclerViewklasa
- RecyclerView.Adapter
- RecyclerView.ViewHolder
- RecyclerZobacz bibliotekę
- Listy w projektowaniu materiałów
- Ulepsz swój interfejs użytkownika dzięki MaterialCardView i obrazom

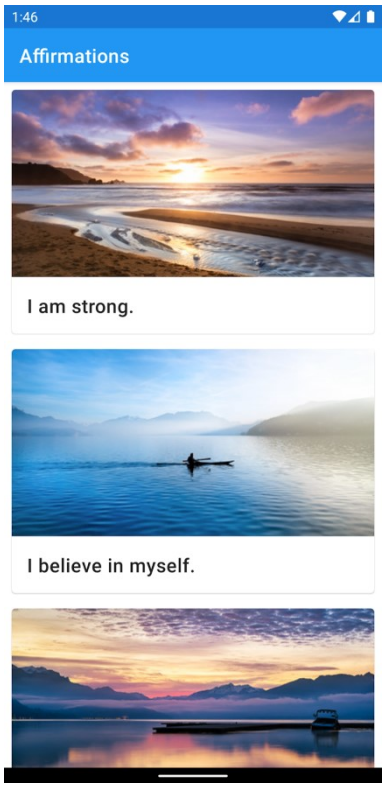
Wyświetl listę obrazów za pomocą kart

1. Zanim zaczniesz

W poprzednim ćwiczeniu z programowania utworzyłeś aplikację Afirmacje , która wyświetla listę tekstów w `RecyclerView`.



W tym kolejnym laboratorium programowania dodajesz inspirujący obraz do każdej afirmacji swojej aplikacji. Wyświetlisz tekst i obraz dla każdej afirmacji na karcie, korzystając z `MaterialCardView`widżetu z biblioteki Material Components for Android. Następnie zakończysz aplikację, dopracowując interfejs użytkownika, aby stworzyć bardziej spójny i piękny interfejs użytkownika. To jest zrzut ekranu ukończonej aplikacji:



Warunki wstępne

- Może dodawać zasoby graficzne do aplikacji.
- Wygodne modyfikowanie układu XML.
- Potrafi stworzyć aplikację, która wyświetla listę tekstów w `RecyclerView`.
- Potrafi stworzyć adapter dla `RecyclerView`.

Czego się nauczysz

- Jak dodać obrazy do listy wyświetlanych afirmacji w `RecyclerView`.
- Jak używać `MaterialCardView`w układzie `RecyclerView`elementów.
- Jak wprowadzić zmiany wizualne w interfejsie użytkownika, aby aplikacja wyglądała na bardziej dopracowaną.

Co zbudujesz

- Dopracowana aplikacja Afirmacje, która używa `RecyclerView`do wyświetlania listy kart. Każda karta zawiera obraz i tekst afirmacji.

Czego potrzebujesz

- Komputer z zainstalowanym Android Studio w wersji 4.1 lub nowszej.
- Dostęp do połączenia internetowego w celu pobierania plików graficznych.
- Aplikacja Afirmacje z poprzedniego ćwiczenia **tworzenia aplikacji Afirmacje** . (Nie podano kodu startowego. Utworzenie aplikacji jest warunkiem wstępnym).

2. Dodawanie obrazów do elementów listy

Do tej pory utworzyłeś adapter `ItemAdapter`do wyświetlania ciągów afirmacji w pliku `RecyclerView`. Funkcjonalnie działa to świetnie, ale wizualnie nie jest zbyt atrakcyjne. W tym zadaniu zmodyfikujesz układ elementu listy i kod adaptera, aby wyświetlać obrazy z afirmacjami.

Pobierz obrazy

1. Aby rozpocząć, otwórz projekt aplikacji Afirmacje w Android Studio z poprzedniego ćwiczenia z programowania. Jeśli nie masz tego projektu, wykonaj kroki opisane w poprzednim ćwiczeniu programowania, aby utworzyć ten projekt. Następnie wróć tutaj.
2. Następnie pobierz pliki graficzne na swój komputer. Powinno być dziesięć obrazów, po jednym dla każdej afirmacji w Twojej aplikacji. Pliki powinny mieć nazwy od `image1.jpg`do `image10.jpg`.
3. Skopiuj obrazy z komputera do folderu **res > drawable** projektu (`app/src/main/res/drawable`) w Android Studio. Gdy te zasoby zostaną dodane do Twojej aplikacji, będziesz mieć dostęp do tych obrazów w swoim kodzie, używając ich identyfikatorów zasobów, takich jak `R.drawable.image1`. (Być może trzeba będzie przebudować kod dla Android Studio, aby znaleźć obraz).

Teraz obrazy są gotowe do użycia w aplikacji.

Dodaj obsługę obrazów w klasie Afirmacja

W tym kroku dodasz właściwość do `Affirmation`klasy danych, która będzie przechowywać wartość identyfikatora zasobu obrazu. W ten sposób `Affirmation`instancja pojedynczego obiektu będzie zawierać identyfikator zasobu dla tekstu afirmacji oraz identyfikator zasobu dla obrazu afirmacji.

1. Otwórz Affirmation.ktplik w modelpakiecie.
2. Zmodyfikuj konstruktora Affirmationklasy, dodając kolejny Intparametr o nazwie imageResourceId.

Korzystanie z adnotacji do zasobów

Oba stringResourceIdi imageResourceIdsą wartościami całkowitymi. Chociaż wygląda to w porządku, wywołujący może przypadkowo przekazać argumenty w złej kolejności (imageResourceIdpierwszy zamiast stringResourceId).

Aby tego uniknąć, możesz użyć adnotacji do zasobów. Adnotacje są przydatne, ponieważ dodają dodatkowe informacje do klas, metod lub parametrów. Adnotacje są zawsze deklarowane z symbolem @. W takim przypadku dodaj @StringResadnotację do właściwości identyfikatora zasobu ciągu i @DrawableResadnotację do właściwości identyfikatora zasobu do rysowania. Następnie otrzymasz ostrzeżenie, jeśli podasz niewłaściwy typ identyfikatora zasobu.

1. Dodaj @StringResadnotację do stringResourceId.
2. Dodaj @DrawableResadnotację do imageResourceId.
3. Upewnij się, że importy androidx.annotation.DrawableResi androidx.annotation.StringRessą dodane na początku pliku po deklaracji pakietu.

Affirmation.kt

```
package com.example.affirmations.model

import androidx.annotation.DrawableRes
import androidx.annotation.StringRes

data class Affirmation(
  @StringRes val stringResourceId: Int,
  @DrawableRes val imageResourceId: Int
)
```

Zainicjuj listę afirmacji obrazami

Teraz, gdy zmieniłeś konstruktora Affirmationklasy, musisz zaktualizować Datasourceklasę. Przekaż identyfikator zasobu obrazu do każdego Affirmationzainicjowanego obiektu.

1. Otwórz Datasource.kt. Powinieneś zobaczyć błąd dla każdego wystąpienia Affirmation.
2. Dla każdego Affirmationdodaj identyfikator zasobu obrazu jako argument, taki jak R.drawable.image1.

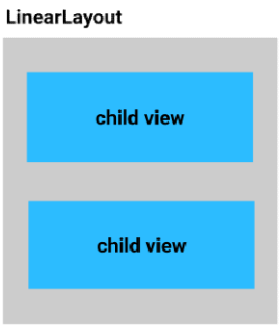
Datasource.kt

```
package com.example.affirmations.data

import com.example.affirmations.R
import com.example.affirmations.model.Affirmation

class Datasource() {

  fun loadAffirmations(): List<Affirmation> {
    return listOf<Affirmation>(
```

```
            Affirmation(R.string.affirmation1, R.drawable.image1),
            Affirmation(R.string.affirmation2, R.drawable.image2),
            Affirmation(R.string.affirmation3, R.drawable.image3),
            Affirmation(R.string.affirmation4, R.drawable.image4),
            Affirmation(R.string.affirmation5, R.drawable.image5),
            Affirmation(R.string.affirmation6, R.drawable.image6),
            Affirmation(R.string.affirmation7, R.drawable.image7),
            Affirmation(R.string.affirmation8, R.drawable.image8),
            Affirmation(R.string.affirmation9, R.drawable.image9),
            Affirmation(R.string.affirmation10, R.drawable.image10)
        )
    }
}
```

Dodaj ImageView do układu elementu listy

Aby wyświetlić obraz dla każdej afirmacji na liście, musisz dodać `ImageView` do układu przedmiotu. Ponieważ masz teraz dwa widoki (a `TextView` i `ImageView`), musisz umieścić je jako widoki potomne w ramach a `ViewGroup`. Aby rozmieścić widoki w pionowej kolumnie, możesz użyć `LinearLayout`. `LinearLayout` wyrównuje wszystkie widoki podrzędne w jednym kierunku, pionowo lub poziomo.



1. Otwórz **res > układ > list_item.xml** . Dodaj `LinearLayout` wokół istniejącego `TextView` i ustaw `orientation` właściwość na `vertical`.
2. Przenieś `xmlns schema` wiersz deklaracji z `TextView` elementu do `LinearLayout` elementu, aby pozbyć się błędu.

`list_item.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <TextView
        android:id="@+id/item_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

```
</LinearLayout>
```

3. Wewnątrz LinearLayout,przed TextView,dodaj an ImageViewz identyfikatorem zasobu item_image.
4. Ustaw ImageViewszerokość na match_parenti wysokość na 194dp. W zależności od rozmiaru ekranu ta wartość powinna pokazywać kilka kart na ekranie w danym momencie.
5. Ustaw scaleTypenacenterCrop.
6. Ustaw importantForAccessibilityatrybut na , noponieważ obraz jest używany do celów dekoracyjnych.

```
<ImageView
  android:layout_width="match_parent"
  android:layout_height="194dp"
  android:id="@+id/item_image"
  android:importantForAccessibility="no"
  android:scaleType="centerCrop" />
```

Zaktualizuj ItemAdapter, aby ustawić obraz

1. Otwórz adapter/ItemAdapter.kt(**aplikacja > java > adapter > ItemAdapter**)
2. Idź do ItemViewHolderklasy.
3. ItemViewHolderInstancja powinna zawierać odwołanie do i TextViewodwołanie do ImageViewukładu elementu listy. Wprowadź następującą zmianę.

   Poniżej inicjalizacji textViewwłaściwości dodaj nazwę o valnazwie imageView. Użyj findViewById(), aby znaleźć odwołanie do ImageViewidentyfikatora item_imagei przypisać je do imageViewwłaściwości.

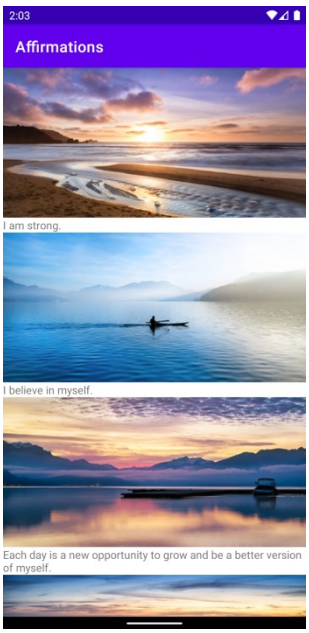
   ItemAdapter.kt

```
class ItemViewHolder(private val view: View): RecyclerView.ViewHolder(view) {
  val textView: TextView = view.findViewById(R.id.item_title)
  val imageView: ImageView = view.findViewById(R.id.item_image)
}
```

4. Znajdź onBindViewHolder()funkcję w ItemAdapter.
5. Wcześniej ustawiłeś afirmację stringResourceIdna textVieww ItemViewHolder. Teraz ustaw element afirmacji imageResourceIdna ImageViewwidok elementu listy.

```
override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
  val item = dataset[position]
  holder.textView.text = context.resources.getString(item.stringResourceId)
  holder.imageView.setImageResource(item.imageResourceId)
}
```

6. Uruchom aplikację i przewiń listę afirmacji.



Aplikacja wygląda znacznie ładniej z obrazami! Jednak nadal możesz ulepszyć interfejs aplikacji. W następnej sekcji dokonasz drobnych zmian w aplikacji, aby ulepszyć interfejs użytkownika.

3. Polerowanie interfejsu użytkownika

Do tej pory zbudowałeś funkcjonalną aplikację, która składa się z listy ciągów afirmacji i obrazów. W tej sekcji zobaczysz, jak drobne zmiany w kodzie i XML mogą sprawić, że aplikacja będzie wyglądać na bardziej dopracowaną.

Dodaj dopełnienie

Na początek dodaj trochę spacji między pozycjami na liście.

Porada: Możesz wprowadzić zmiany układu w pliku XML, jak pokazano tutaj, lub możesz je wprowadzić w panelu **Atrybuty** w widoku **Projekt** , w zależności od preferencji.

1. Otwórz list_item.xml(**app** > **res** > **layout** > **item_list.xml**) i dodaj 16dpdopełnienie do istniejącego LinearLayout.

list_item.xml

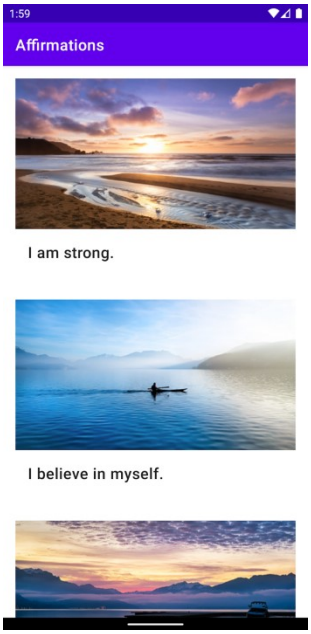
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:orientation="vertical"
  android:padding="16dp">
```

2. Dodaj 16dpdopełnienie do item_title TextView.
3. W TextView, ustaw textAppearanceatrybut na ?attr/textAppearanceHeadline6. textAppearanceto atrybut, który pozwala na zdefiniowanie stylu specyficznego dla tekstu. Inne możliwe wstępnie zdefiniowane

wartości wyglądu tekstu można znaleźć w sekcji TextAppearances w tym wpisie na blogu dotyczącym wspólnych atrybutów motywu .

```
<TextView
  android:id="@+id/item_title"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:padding="16dp"
  android:textAppearance="?attr/textAppearanceHeadline6" />
```

4. Uruchom aplikację. Czy uważasz, że lista wygląda lepiej?



Użyj kart

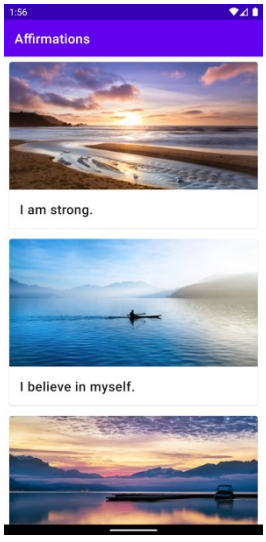
Nadal trudno jest stwierdzić, czy obraz należy do tekstu afirmacji nad lub pod tym obrazem. Aby to naprawić, możesz użyć widoku **karty** . Widok karty zapewnia łatwy sposób przechowywania grupy widoków, zapewniając jednocześnie spójny styl kontenera. Aby uzyskać więcej wskazówek dotyczących korzystania z kart w Material Design, zapoznaj się z tym przewodnikiem dotyczącym kart .

1. Dodaj `MaterialCardView`wokół istniejącego `LinearLayout`.
2. Ponownie przenieś deklarację schematu z `LinearLayout`do `MaterialCardView`.
3. Ustaw `layout_width` na `MaterialCardView`, `match_parent`i `layout_height`na `wrap_content`.
4. Dodaj `layout_margin`z `8dp`.
5. Usuń dopełnienie w `LinearLayout`, aby nie mieć za dużo spacji.
6. Teraz ponownie uruchom aplikację. Czy możesz lepiej odróżnić każdą afirmację od `MaterialCardView`?

`list_item.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.material.card.MaterialCardView
xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
```

```
    android:layout_margin="8dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <ImageView
            android:id="@+id/item_image"
            android:layout_width="match_parent"
            android:layout_height="194dp"
            android:importantForAccessibility="no"
            android:scaleType="centerCrop" />

        <TextView
            android:id="@+id/item_title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="16dp"
            android:textAppearance="?attr/textAppearanceHeadline6" />

    </LinearLayout>

</com.google.android.material.card.MaterialCardView>
```



Zmień kolory motywu aplikacji

Domyślny kolor motywu aplikacji może nie być tak uspokajający, jak niektóre inne wybory, których możesz dokonać. W tym zadaniu zmienisz kolor motywu aplikacji na niebieski. Następnie możesz to zmienić ponownie, korzystając z własnych pomysłów!

Predefiniowane odcienie niebieskiego z palety kolorów Material Design można znaleźć pod tym linkiem .

W tym ćwiczeniu z programowania będziesz używać następujących kolorów z palety Material Design:

- niebieski_200:`#FF90CAF9`
- niebieski_500:`#FF2196F3`
- niebieski_700:`#FF1976D2`

Dodaj zasoby kolorów

Zdefiniuj kolory używane w Twojej aplikacji w centralnym miejscu: `colors.xml`pliku.

1. Otwórz `colors.xml`(**res > kolor > kolory.xml**).
2. Dodaj nowe zasoby kolorów do pliku dla kolorów niebieskich zdefiniowanych poniżej:

```
<color name="blue_200">#FF90CAF9</color>
<color name="blue_500">#FF2196F3</color>
<color name="blue_700">#FF1976D2</color>
```

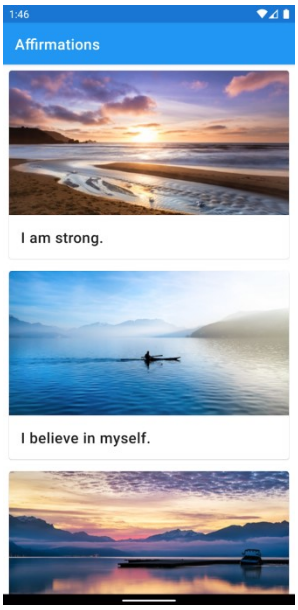
Zmień kolory motywu

Teraz, gdy masz nowe zasoby kolorów, możesz ich użyć w swoim motywie.

1. Otwórz `themes.xml`(**res > wartości > motywy > motywy.xml**).
2. Znajdź `<!-- Primary brand color. -->`sekcję.
3. Dodaj lub zmień `colorPrimary`, aby użyć `@color/blue_500`.
4. Dodaj lub zmień `colorPrimaryVariant`, aby użyć `@color/blue_700`.

```
<item name="colorPrimary">@color/blue_500</item>
<item name="colorPrimaryVariant">@color/blue_700</item>
```

5. Uruchom aplikację. Powinieneś zobaczyć, że kolor paska aplikacji zmienił się na niebieski.



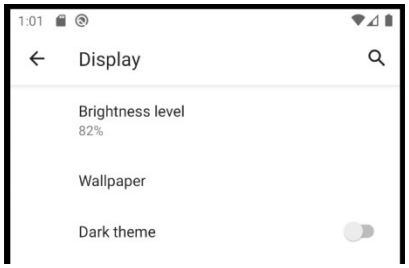
Zaktualizuj ciemne kolory motywu

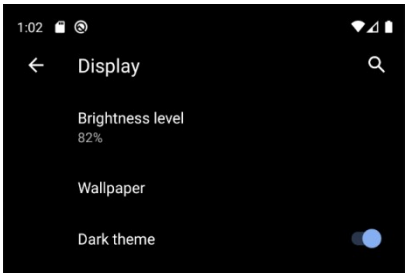
Dla ciemnego motywu aplikacji dobrze jest wybrać bardziej nasycone kolory .

1. themes.xmlOtwórz plik ciemnego motywu (**motywy > motywy.xml (noc)**).
2. Dodaj lub zmień atrybuty colorPrimaryi motywu w następujący sposób:colorPrimaryVariant

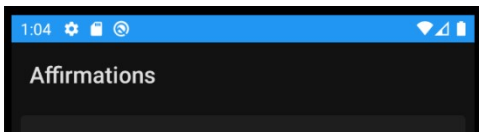
```
<item name="colorPrimary">@color/blue_200</item>
<item name="colorPrimaryVariant">@color/blue_500</item>
```

3. Uruchom swoją aplikację.
4. W **Ustawieniach** urządzenia włącz **Ciemny motyw** .





5. Twoja aplikacja przełączy się na **ciemny motyw.** Sprawdź, czy wygląda jak na poniższym zrzucie ekranu:



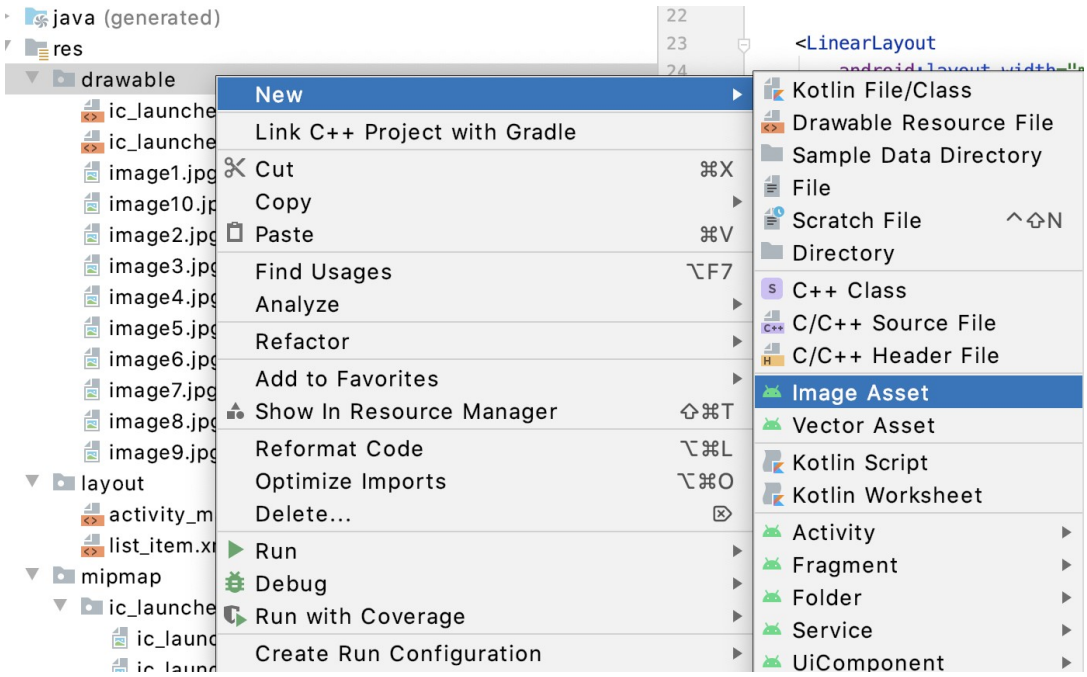
Uwaga: kolor paska aplikacji**.** Być może zastanawiasz się, dlaczego określony kolor podstawowy dla ciemnego motywu nie jest wyświetlany na pasku aplikacji. Ciemny pasek aplikacji jest zgodny z projektem. W ciemnym motywie pasek aplikacji i inne duże obszary są domyślnie wyświetlane z ciemnym tłem (colorSurface) zamiast koloru podstawowego. Dzieje się tak, ponieważ ciemny motyw Material zaleca mniejsze użycie jasnych kolorów na dużych powierzchniach. Przyciski lub inne małe akcenty pokażą zdefiniowany kolor podstawowy.

6. W tym momencie możesz również usunąć nieużywane kolory z colors.xmlpliku (na przykład zasoby koloru fioletowego używane w domyślnym motywie aplikacji).

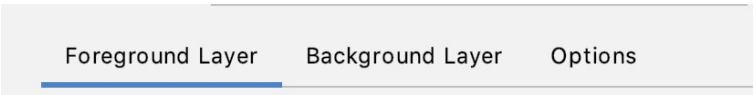
Zmień ikonę aplikacji

Na koniec zaktualizujesz ikonę aplikacji.

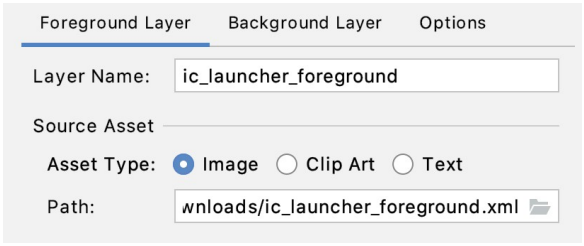
1. Pobierz pliki ikon aplikacji ic_launcher_foreground.xmli ic_launcher_background.xml. Jeśli przeglądarka wyświetla plik zamiast go pobierać, wybierz **Plik > Zapisz stronę jako...** , aby zapisać go na komputerze.
2. W Android Studio usuń dwa pliki: drawable/ic_launcher_background.xmli drawable-v24/ic_launcher_foreground.xmlpliki, ponieważ dotyczą one poprzedniej ikony aplikacji. Możesz odznaczyć pole **Bezpieczne usuwanie (z wyszukiwaniem użycia)** .
3. Następnie kliknij prawym przyciskiem myszy **res >** folder do rysowania i wybierz **Nowy > Zasób obrazu** .



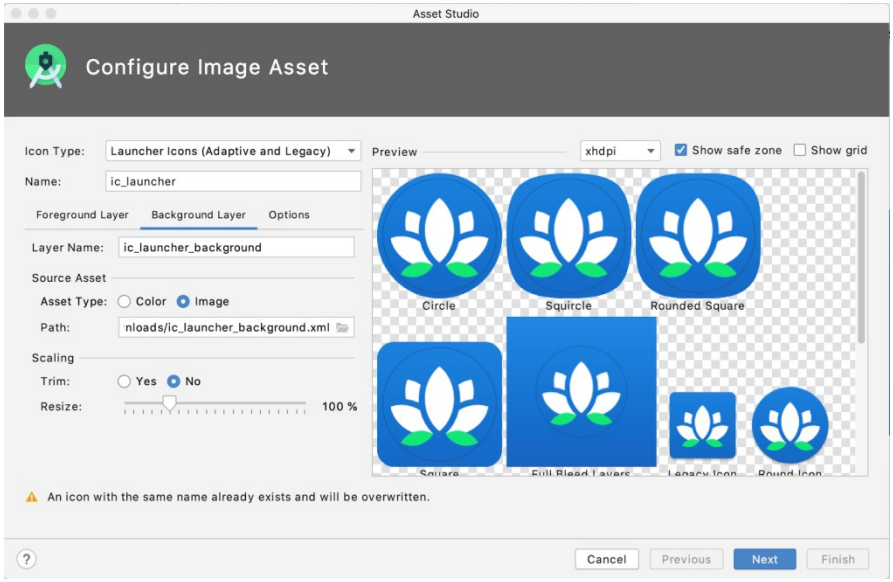
4. W oknie **Konfiguruj zasób obrazu** upewnij się, że wybrana jest **warstwa pierwszego planu .**



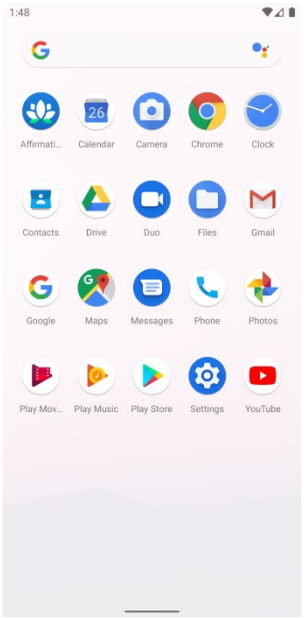
5. Poniżej znajdź etykietę **Ścieżka .**
6. Kliknij ikonę folderu w polu tekstowym **Ścieżka .**
7. Znajdź i otwórz `ic_launcher_foreground.xml`plik pobrany na komputer.



8. Przejdź do zakładki **Warstwa tła .**
9. Kliknij ikonę **Przeglądaj** w polu tekstowym **Ścieżka .**
10. Znajdź i otwórz `ic_launcher_background.xml`plik na swoim komputerze. Żadne inne zmiany nie są konieczne.
11. Kliknij **Dalej** .

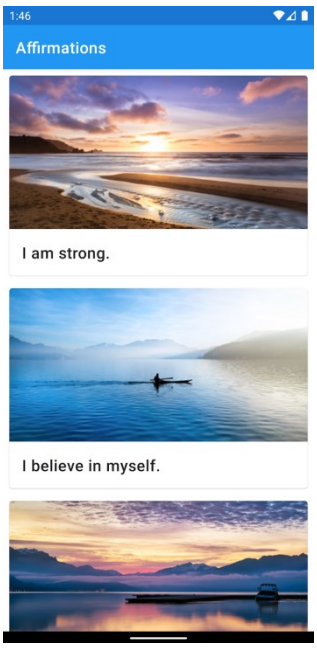


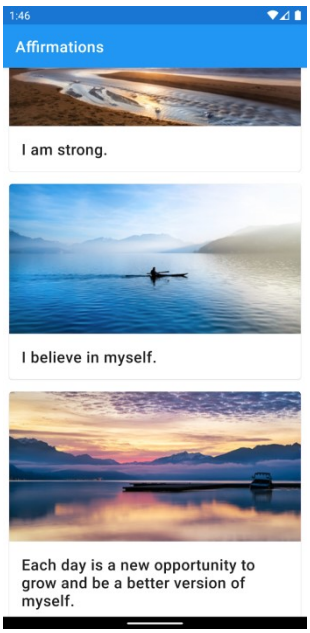
12. W oknie dialogowym **Potwierdź ścieżkę ikony** kliknij przycisk **Zakończ** . Można nadpisać istniejące ikony.
13. Aby uzyskać najlepsze praktyki, możesz przenieść nowe elementy wektorowe do rysowania `ic_launcher_foreground.xml`do `ic_launcher_background.xml`nowego katalogu zasobów o nazwie `drawable-anydpi-v26`. Ikony adaptacyjne zostały wprowadzone w API 26, więc te zasoby będą używane tylko na urządzeniach z API 26 i wyższym (dla dowolnego dpi).
14. Usuń `drawable-v24`katalog, jeśli nic tam nie zostało.
15. Uruchom swoją aplikację i zwróć uwagę na piękną nową ikonę aplikacji w szufladzie aplikacji!



16. Jako ostatni krok nie zapomnij ponownie sformatować plików Kotlin i XML w projekcie, aby Twój kod był czystszy i przestrzegał wytycznych dotyczących stylu.

    Gratulacje! Stworzyłeś inspirującą aplikację Afirmacje.





Korzystając z tej wiedzy o tym, jak wyświetlić listę danych w aplikacji na Androida, co możesz zbudować dalej?

4. Kod rozwiązania

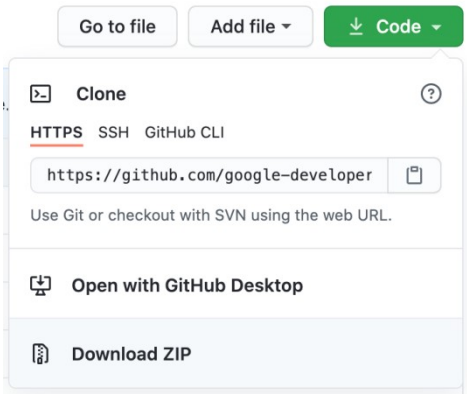
Kod rozwiązania dla aplikacji Afirmacje znajduje się w repozytorium GitHub poniżej:

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

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

Pobierz kod

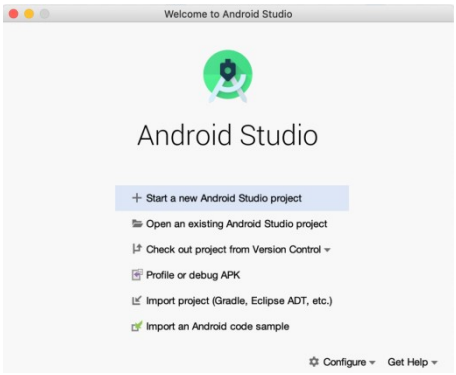
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



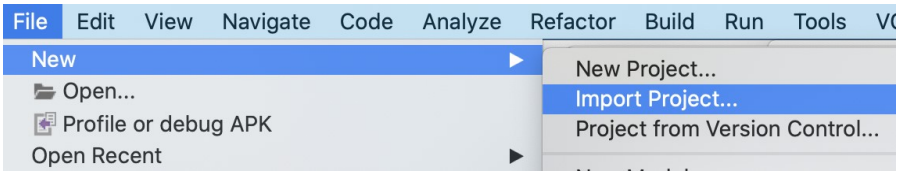
3. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
4. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
5. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego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

5. Podsumowanie

- Aby wyświetlić dodatkową zawartość w RecyclerView, zmodyfikuj podstawową klasę modelu danych i źródło danych. Następnie zaktualizuj układ elementu listy i adapter, aby ustawić te dane w widokach.
- Użyj adnotacji do zasobów, aby upewnić się, że właściwy typ identyfikatora zasobu jest przekazywany do konstruktora klasy.
- Użyj **biblioteki Material Components for Android,** aby aplikacja była łatwiejsza do przestrzegania zalecanych wytycznych dotyczących projektowania materiałów.
- Użyj MaterialCardView, aby wyświetlić zawartość na karcie materiału.
- Niewielkie wizualne poprawki w aplikacji pod względem kolorów i odstępów mogą sprawić, że aplikacja będzie wyglądać bardziej dopracowana i spójna.

6. Dowiedz się więcej

- Utwórz listę za pomocąRecyclerView
- RecyclerViewklasa
- RecyclerViewAdaptery
- RecyclerViewViewHolder
- Listy w projektowaniu materiałów
- Karty w projektowaniu materiałów
- MaterialCardView
- Pierwsze kroki z komponentami materiałowymi dla Androida
- Stylizacja na Androida: motywy a style
- Ikony adaptacyjne

7. 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.

W tej serii ćwiczeń z programowania nauczyłeś się używać LinearLayoutManagerz RecyclerView. RecyclerViewmoże używać różnych menedżerów LayoutManager, aby układać dane w różny sposób.

- Zmień layoutManagerwłaściwość RecyclerViewna GridLayoutManager.
- Zmień liczbę kolumn na 3.
- Zmień układ adaptera, aby zwizualizować dane w siatce.

Listy testowe i adaptery

1. Zanim zaczniesz

W poprzednich ćwiczeniach z kodowania nauczyłeś się pisać i uruchamiać zarówno testy jednostkowe, jak i oprzyrządowanie. W tym ćwiczeniu z programowania przedstawiono kilka najlepszych praktyk podczas pisania testów oraz jak dodać określone zależności Gradle do testowania. Będziesz także mógł przećwiczyć pisanie większej liczby testów jednostkowych i oprzyrządowania.

Warunki wstępne

- Otworzyłeś istniejący projekt w Android Studio.
- Napisałeś testy jednostkowe i oprzyrządowanie w Android Studio.
- Masz pewne doświadczenie w nawigowaniu projektami w Android Studio.
- Masz pewne doświadczenie w pracy z `build.gradle`plikami w Android Studio.

Czego się nauczysz

- Podstawy pisania testu.
- Jak dodać zależności Gradle specyficzne dla testowania.
- Jak testować listy za pomocą testów oprzyrządowania.

Czego potrzebujesz

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

Pobierz kod startowy do tego ćwiczenia z programowania

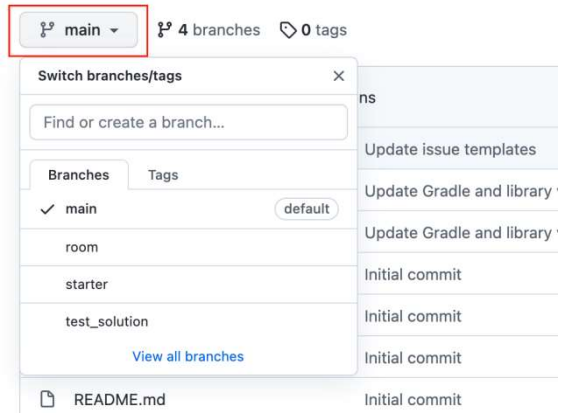
W tym laboratorium programowania dodasz testy instrumentacji do kodu rozwiązania dla aplikacji **Afirmacje** .

URL kodu startowego: https://github.com/google-developer-training/android-basics-kotlin-affirmations-app-solution

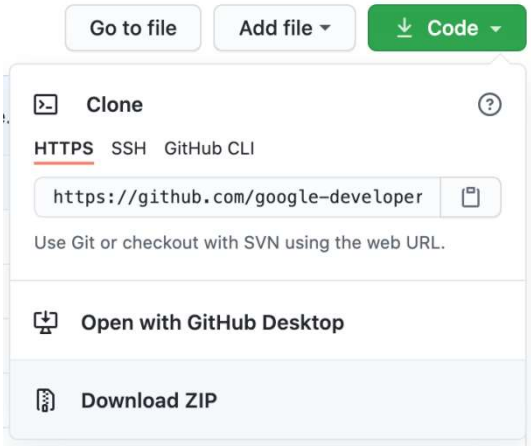
Nazwa modułu z kodem startowym:

`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** .



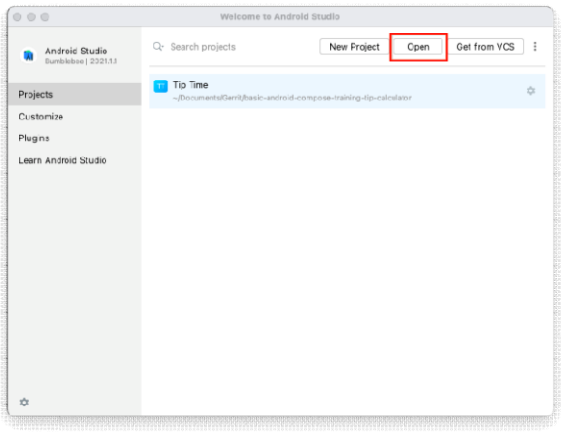
3. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli wyskakujące okienko.



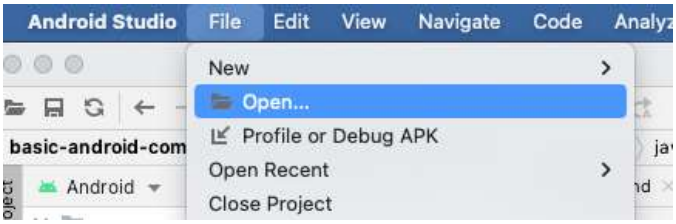
4. W wyskakującym okienku kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na komputerze. Poczekaj na zakończenie pobierania.
5. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
6. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego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

2. Przegląd aplikacji startowej

Aplikacja **Afirmacje** składa się z jednego ekranu, który pokazuje użytkownikowi listę obrazów połączonych ze słowami afirmacji.

3. Najlepsze praktyki

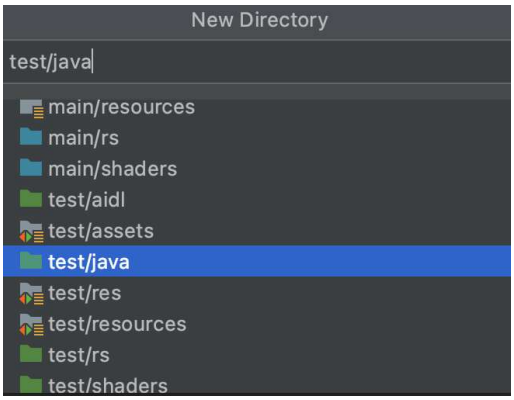
Z założenia kod testowy różni się od logiki biznesowej aplikacji. To dlatego, że testy nie powinny zawierać logiki; mają to tylko przetestować. Dlatego testy nie powinny zawierać instrukcji warunkowych, takich jak `if`lub `when`, ani instrukcji przepływu sterowania, takich jak `for`lub `while`. Nie powinni również manipulować wartościami ani przeprowadzać żadnych prawdziwych obliczeń.

Czasami twoje testy mogą wymagać niektórych z tych rzeczy, ale generalnie powinieneś ich unikać. Ponieważ jest to typ logiki, który chcemy przetestować w naszej aplikacji, gdybyśmy mieli taki rodzaj kodu w teście, mógłby zakończyć się niepowodzeniem w taki sam sposób, jak nasz kod aplikacji.

Nasze testy jednostkowe powinny wywoływać tylko fragment kodu z naszej aplikacji, który jest niezbędny do testu i testować wartości lub stan kodu, który wynika z wywołania tego kodu. Testy interfejsu użytkownika powinny sprawdzać tylko oczekiwany stan interfejsu użytkownika. Ta koncepcja może trochę potrwać, ale to w porządku! Istnieje kilka tematów, które pomogą wyjaśnić tę koncepcję, które omówimy w przyszłych ćwiczeniach z programowania. W międzyczasie, gdy piszemy więcej testów, zwracaj szczególną uwagę na podejście, jakie stosujemy do pisania testów.

4. Utwórz katalogi testów

W poprzednim ćwiczeniu z kodowania nauczyłeś się tworzyć androidTestkatalog do testów instrumentacji. Powtórz ten proces dla tego projektu zarówno dla androidTestkatalogu, jak i testkatalogu. Proces jest taki sam dla obu, jedyną różnicą jest to, że w przypadku testkatalogu musisz wybrać **test/java** z menu rozwijanego **Nowy katalog** zamiast **androidTest/java** . Utwórz nowy pakiet dla każdego nowego katalogu o nazwie **com.example.affirmations** .



5. Utwórz klasę testową oprzyrządowania

Utwórz nową klasę w **androidTest -> com.example.affirmations** o nazwie AffirmationsListTests.kt.

Podobnie jak w przypadku aplikacji **Dice Roller** , **afirmacje** mają tylko jedną czynność. Aby przetestować interfejs użytkownika aktywności, musimy określić, że chcemy go uruchomić. Sprawdź, czy potrafisz sobie przypomnieć, jak to zrobić samodzielnie!

1. Dodaj biegacza testowego do nowo utworzonej klasy.

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

2. Utwórz regułę scenariusza działań dla działania głównego.

   ```
   @get:Rule
   val activity = ActivityScenarioRule(MainActivity::class.java)
   ```

3. **Afirmacje** wyświetla listę obrazów i ich odpowiednich pozytywnych afirmacji. Interfejs użytkownika nie pozwala na żadną interakcję z elementami (na przykład klikanie lub przesuwanie). W przypadku tej aplikacji test oprzyrządowania sprawdza tylko dane statyczne. Utwórz metodę testową o nazwie scroll_to_item(). Pamiętaj, że musi mieć adnotację @Test.

   Ten test powinien przewinąć do określonej pozycji zawartej na liście. Nie omówiliśmy jeszcze tego podejścia, ponieważ wymaga ono metody, do której nasz projekt nie ma jeszcze odniesienia. Przed kontynuowaniem testu musimy dodać kilka zależności testowych.

6. Dodawanie zależności testowych oprzyrządowania

Powinieneś już mieć pewną wiedzę na temat dodawania zależności Gradle do użycia w kodzie aplikacji. Gradle pozwala nam również dodawać zależności specjalnie dla testów jednostkowych i testów oprzyrządowania. build.gradleOtwórz plik poziomu aplikacji znajdujący się w **app -> build.gradle** . W sekcji zależności istnieją trzy rodzaje implementacji zależności: implementation, testImplementation,i androidTestImplementation.

implementationdotyczy zależności, które będą używane w samej aplikacji, testImplementationdotyczy zależności używanych w testach jednostkowych i androidTestImplementationjest dla zależności używanych w testach instrumentacji.

1. Dodaj zależność, aby umożliwić interakcję z RecyclerView's w testach instrumentacji. Dodaj następującą bibliotekę jako androidTestImplementation:

   androidx.test.espresso:espresso-contrib:3.4.0

   Uwaga: najnowszą wersję można znaleźć espresso-contrib tutaj .

   Zależności wyglądają mniej więcej tak:

   ```
   dependencies {
     ...
     androidTestImplementation
   'androidx.test.espresso:espresso-contrib:3.4.0'
   }
   ```

2. Teraz zsynchronizuj projekt.

7. Przetestuj RecyclerView

1. Po zsynchronizowaniu projektu wróć do AffirmationsListTests.ktpliku. Podaj a, ViewInteractionaby wykonać czynność na onView(). Metoda onView()wymaga podania a ViewMatcher. Użyj withId(), upewniając się, że przekazałeś identyfikator, RecyclerViewktóry został użyty do afirmacji. Teraz zadzwoń perform()na ViewInteraction. W tym miejscu w grę wchodzi nowo dodana zależność! RecyclerViewActions.scrollToPosition<RecyclerView.Viewholder>(9) ViewActionTeraz można przekazać .

   ```
   onView(withId(R.id.recycler_view)).perform(
     RecyclerViewActions
       .scrollToPosition<RecyclerView.ViewHolder>(9))
   )
   ```

   Zrozumienie składni tego wiersza nie jest krytyczne, ale warto to zbadać. Nazwa RecyclerViewActionsjest dokładnie tym, co sugeruje nazwa: klasa, która pozwala twoim testom wykonywać akcje na RecyclerView. scrollToPosition()jest statyczną metodą z RecyclerViewActionsklasy, która przewinie się do określonej pozycji. Ta metoda zwraca to, co nazywa się Generic. Generyki są poza zakresem tego ćwiczenia z programowania, ale w tym przypadku możesz myśleć o tym jako o scrollToPosition()metodzie zwracającej dowolne elementy w obszarze RecyclerView, które mogą być dowolne.

W naszej aplikacji elementy w naszym RecyclerViewsą ViewHolders, więc po wywołaniu metody umieszczamy parę nawiasów ostrych i w nich określamy RecyclerView.ViewHolder. Na koniec przejdź na ostatnią pozycję na liście (9).

Uwaga: Zwróć uwagę, że dla pozycji została przekazana zakodowana na stałe wartość 9, działa to tylko dlatego, że rozmiar listy w tej aplikacji jest statyczny. Jeśli aplikacja miała dynamiczny rozmiar listy, użycie wartości zakodowanej na sztywno nie byłoby dobrą praktyką, ponieważ rozmiar jest nieznany, a RecyclerViewadapter z MainActivitynie może być dostępny, ponieważ zmienna przechowująca RecyclerViewjest zawarta w onCreate()metodzie. Jeśli zmienna została zadeklarowana na poziomie klasy MainActivity, moglibyśmy uzyskać do niej dostęp, aby uzyskać rozmiar adaptera. Ponadto testy interfejsu użytkownika niekoniecznie powinny uzyskiwać dostęp do zmiennych w testowanej aktywności; takie podejście byłoby bardziej odpowiednie dla testu jednostkowego.

Unikanie wartości zakodowanych na stałe w takim teście zostanie omówione w kroku 2 poniżej.

2. Teraz, gdy przewijanie do żądanej pozycji RecyclerViewjest włączone, wykonaj potwierdzenie, aby upewnić się, że interfejs użytkownika wyświetla oczekiwane informacje. Upewnij się, że po przewinięciu do ostatniej pozycji wyświetlony zostanie tekst związany z ostateczną afirmacją. Zacznij od ViewInteraction, ale tym razem przekaż nowy ViewMatcher(w tym przypadku withText()). Do tej metody przekaż zasób ciągu, który zawiera tekst ostatniej afirmacji. Metoda withText()identyfikuje składnik interfejsu użytkownika na podstawie wyświetlanego tekstu. W przypadku tego komponentu wszystko, co należy zrobić, to sprawdzić, czy wyświetla żądany tekst. Odbywa się to poprzez check()wywołanie ViewInteraction. check()wymaga ViewAssertion, dla którego możesz użyćmatches()metoda. Na koniec zrób potwierdzenie, że składnik interfejsu użytkownika jest wyświetlany, przekazując method isDisplayed().

```
onView(withText(R.string.affirmation10))
  .check(matches(isDisplayed()))
```

Wracając do uwagi o twardym kodowaniu pozycji do przewijania, istnieje sposób na pokonanie tego za pomocą RecyclerViewActions. Jeśli nie masz pewności co do długości swojej listy, możesz użyć scrollToakcji. Funkcja scrollTowymaga a Matcher<View!>!do znalezienia konkretnego elementu. Może to być kilka rzeczy, ale aby służyć celom tego testu, użyj withText. Stosując to do testu, który właśnie napisałeś, kod będzie wyglądał tak:

```
onView(withId(R.id.recycler_view)).perform(
  RecyclerViewActions
    .scrollTo<RecyclerView.ViewHolder>(
      withText(R.string.affirmation10)
    )
)

onView(withText(R.string.affirmation10))
  .check(matches(isDisplayed())
)
```

Uwaga: Istnieje wiele przydatnych funkcji związanych z klasą RecyclerViewActions, o których możesz przeczytać tutaj.

Teraz wszystko jest gotowe do uruchomienia testu. Powinieneś zobaczyć, że urządzenie lub emulator przewinęło się na dół listy, a następnie test przeszedł. Jeśli chcesz się upewnić, że wynik testu jest

dokładny, zamień identyfikator ciągu na `R.string.affirmation1`. Po przewinięciu ten zasób ciągu nie jest wyświetlany i test powinien zakończyć się niepowodzeniem.

W `RecyclerViewActions`klasie dostępnych jest kilka metod, zachęcamy do zapoznania się z dostępnymi metodami .

8. Utwórz lokalną klasę testową

Utwórz nową klasę w **test -> com.example.affirmations** o nazwie `AffirmationsAdapterTests.kt`.

9. Dodawanie lokalnych zależności testowych

1. Wcześniej w tym laboratorium kodowania omówiliśmy trzy różne typy implementacji zależności i dodano zależność dla testów instrumentacji. Teraz dodaj zależność dla testów lokalnych. Przejdź do **app -> build.gradle** i dodaj następujące jako zależność testu jednostkowego:

   ```
   org.mockito:mockito-core:3.12.4
   ```

   Zależności powinny wyglądać mniej więcej tak:

   ```
   dependencies {
     ...
     testImplementation 'org.mockito:mockito-core:3.12.4'
   }
   ```

2. Teraz zsynchronizuj projekt.

10. Przetestuj adapter

Ta konkretna aplikacja nie nadaje się do testów jednostkowych, ponieważ nie ma zbyt wiele logiki do przetestowania. Możemy jednak zdobyć trochę więcej doświadczenia w testowaniu różnych komponentów jako przygotowanie do przyszłych testów.

1. Umieść następujący wiersz w klasie testu jednostkowego:

   ```
   private val context = mock(Context::class.java)
   ```

   Metoda `mock()`pochodzi z biblioteki, którą właśnie wdrożyliśmy w naszym projekcie. Mockowanie jest integralną częścią testów jednostkowych, ale nie wchodzi w zakres tego ćwiczenia z kodowania. Prześmiewstwo omówimy bardziej szczegółowo w osobnym ćwiczeniu z programowania. W Androidzie `Context`jest to kontekst aktualnego stanu aplikacji, ale pamiętaj, że testy jednostkowe są uruchamiane na JVM, a nie na rzeczywistym urządzeniu, więc nie ma `Context`. Metoda mock pozwala nam stworzyć "wyśmiewaną" instancję `Context`. Nie ma żadnej rzeczywistej funkcjonalności, ale może być używany do testowania metod wymagających kontekstu.

2. Utwórz wywołaną funkcję adapter_size()i opisz ją jako test. Celem tego testu jest upewnienie się, że rozmiar adaptera odpowiada rozmiarowi listy przekazanej do adaptera. Aby to zrobić, utwórz instancję ItemAdapteri przekaż listę zwróconą przez loadAffirmations()metodę w Datasourceklasie. Alternatywnie utwórz nową listę i przetestuj ją. W przypadku testów jednostkowych najlepszym rozwiązaniem jest utworzenie własnych danych unikalnych dla testu, dlatego utworzymy listę niestandardową dla tego testu.

```
val data = listOf(
  Affirmation(R.string.affirmation1, R.drawable.image1),
  Affirmation(R.string.affirmation2, R.drawable.image2)
)
```

3. Teraz utwórz instancję ItemAdapter, przekazując zmienne contexti datautworzone w poprzednich krokach.

```
val adapter = ItemAdapter(context, data)
```

Adaptery widoku Recyklera mają metodę zwracającą rozmiar adaptera o nazwie getItemCount(). W przypadku tej aplikacji metoda wygląda tak:

```
/**
* Return the size of your dataset (invoked by the layout manager)
*/
override fun getItemCount() = dataset.size
```

4. To jest metoda, którą należy przetestować. Upewnij się, że wartość zwrócona z tej metody jest zgodna z rozmiarem listy utworzonej w kroku 2. Użyj assertEquals()metody i porównaj wartości rozmiaru listy i rozmiaru adaptera.

```
assertEquals("ItemAdapter is not the correct size", data.size, adapter.itemCount)
```

Znasz już tę assertEquals()metodę, ale warto przyjrzeć się linijce, aby była dokładna. Pierwszy parametr to ciąg znaków, który wyświetla się w wyniku testu, jeśli test się nie powiedzie. Drugi parametr to wartość oczekiwana. Trzeci parametr to rzeczywista wartość. Twoja klasa testowa powinna wyglądać tak:

```kotlin
package com.example.affirmations

import android.content.Context
import com.example.affirmations.adapter.ItemAdapter
import com.example.affirmations.model.Affirmation
import org.junit.Assert.assertEquals
import org.junit.Test
import org.mockito.Mockito.mock

class AffirmationsAdapterTests {

    private val context = mock(Context::class.java)

    @Test
    fun adapter_size() {
        val data = listOf(
            Affirmation(R.string.affirmation1, R.drawable.image1),
            Affirmation(R.string.affirmation2, R.drawable.image2)
        )
        val adapter = ItemAdapter(context, data)
        assertEquals( message: "ItemAdapter is not the correct size", data.size, adapter.itemCount)
    }
}
```

5. Teraz uruchom test!

11. Kod rozwiązania

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

Nazwa modułu z kodem rozwiązania:

test_solution

12. Gratulacje

W tym ćwiczeniu z programowania możesz:

- Dowiedz się, jak dodawać zależności specyficzne dla testów.
- Dowiedz się, jak współdziałać z RecyclerViewtestami oprzyrządowania.
- Omówiono podstawowe najlepsze praktyki testowania.

Projekt: Aplikacja Dogglers

1. Zanim zaczniesz

To laboratorium programowania przedstawia nową aplikację o nazwie Dogglers, którą zbudujesz samodzielnie. To laboratorium kodowania przeprowadzi Cię przez kroki, aby ukończyć projekt aplikacji Dogglers, 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 poprzedniego kodu rozwiązania z innych aplikacji w jednostce 2) z kodem, który piszesz.

Na początku może się to wydawać zniechęcające, ale jesteśmy w 100 procentach pewni, że jeśli udało ci się ukończyć część 2, 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 2 kursu Android Basics in Kotlin.

Co zbudujesz

- Zbudujesz aplikację o nazwie Dogglers, która wyświetla informacje w widoku RecyclerView, korzystając z umiejętności, których nauczyłeś się w części 2.

Co będziesz potrzebował

- Komputer z zainstalowanym Android Studio.

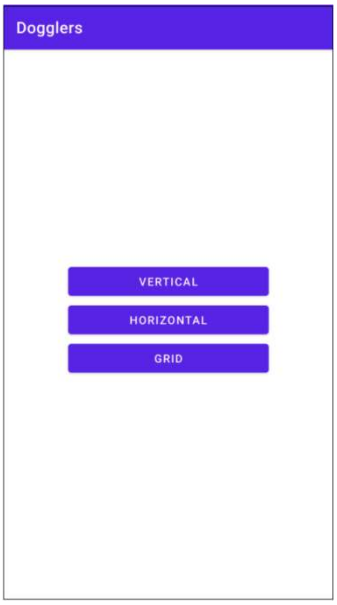
2. Przegląd aplikacji

Witamy w aplikacji Project: Dogglers!

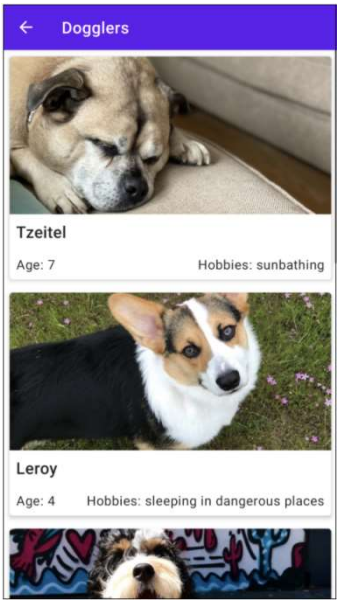
W Google czule nazywamy naszych współpracowników Googlersami. Ponieważ wielu Googlersów ma psy jako zwierzęta domowe, pomyśleliśmy, że fajnie byłoby stworzyć aplikację dla naszych psich przyjaciół,

zwaną Dogglers. Twoim zadaniem jest zaimplementowanie Dogglers, które wyświetlają przewijane listy psów domowych Googlerów wraz z odrobiną informacji o każdym z nich, takich jak imię, wiek, hobby i zdjęcie. W tym projekcie zbudujesz układy dla elementów RecyclerView w aplikacji Dogglers i zaimplementujesz adapter, aby lista psów mogła być prezentowana na trzy sposoby: przez przewijanie w poziomie, przewijanie w pionie i układ siatki z przewijaniem w pionie.

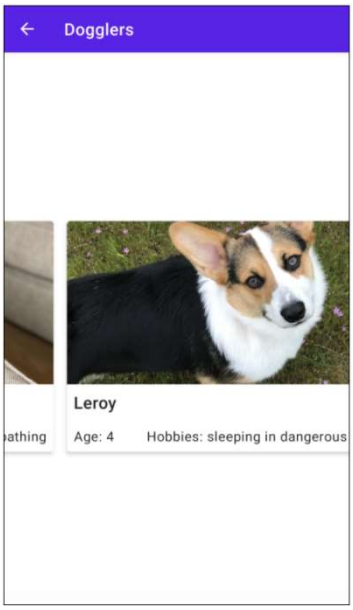
Po uruchomieniu aplikacji dostępne są opcje układu poziomego, pionowego i siatki.



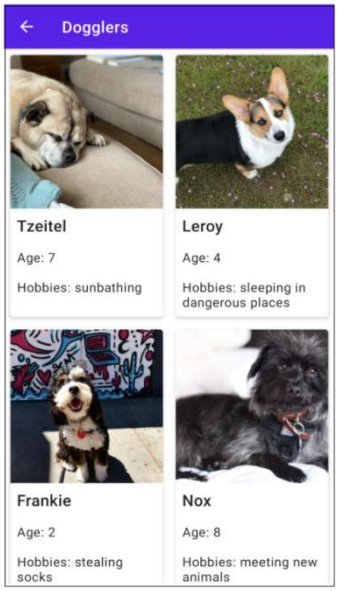
Pierwsza opcja to pionowy widok recyklera przewijania z elementami zajmującymi całą szerokość ekranu.



Druga opcja pokazuje listę psów w widoku recyklera z przewijaniem w poziomie.



Trzecia opcja pokazuje psy w układzie pionowym przewijanym w stylu siatki, gdzie dwa psy są pokazane w każdym rzędzie.



Wszystkie te układy są zasilane przez tę samą klasę adaptera. Twoim zadaniem będzie zbudowanie układów kart widoku recyklera, a następnie zaimplementowanie adaptera, tak aby każdy element był wypełniony informacjami o każdym psie.

3. Rozpocznij

Pobierz kod projektu

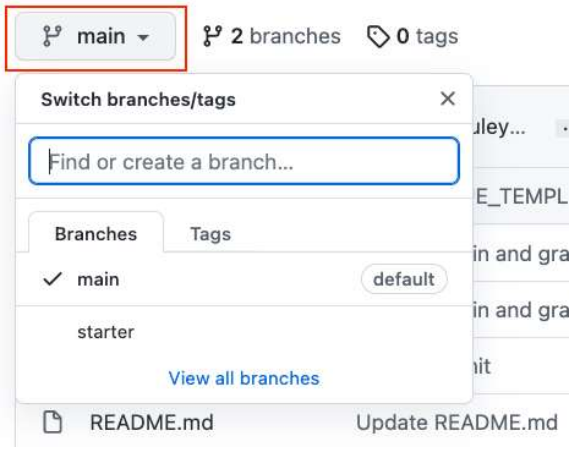
Zauważ, że nazwa folderu to `android-basics-kotlin-dogglers-app`. Wybierz ten folder podczas otwierania projektu w Android Studio.

Adres URL kodu startowego:

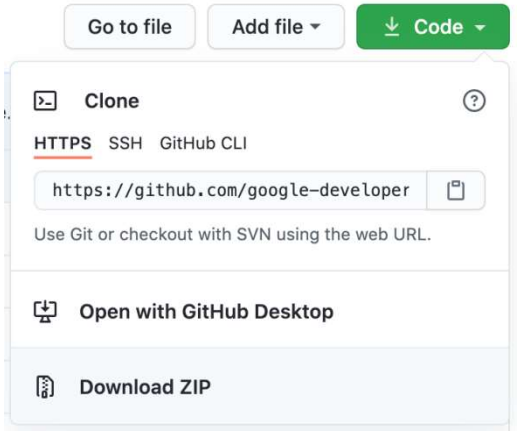
https://github.com/google-developer-training/android-basics-kotlin-dogglers-app/tree/main

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** .



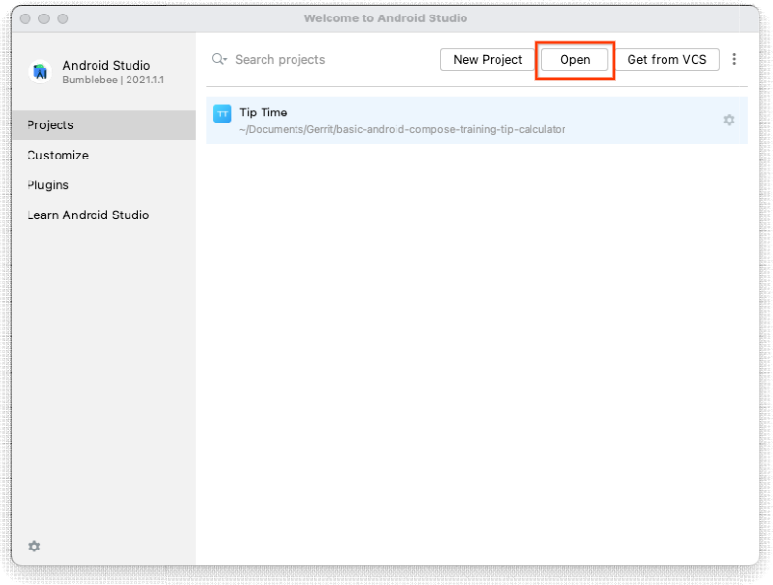
3. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli wyskakujące okienko.



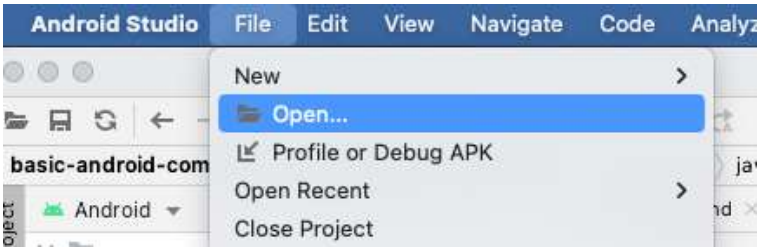
4. W wyskakującym okienku kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na komputerze. Poczekaj na zakończenie pobierania.
5. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
6. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego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

Otwórz projekt w Android Studio

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

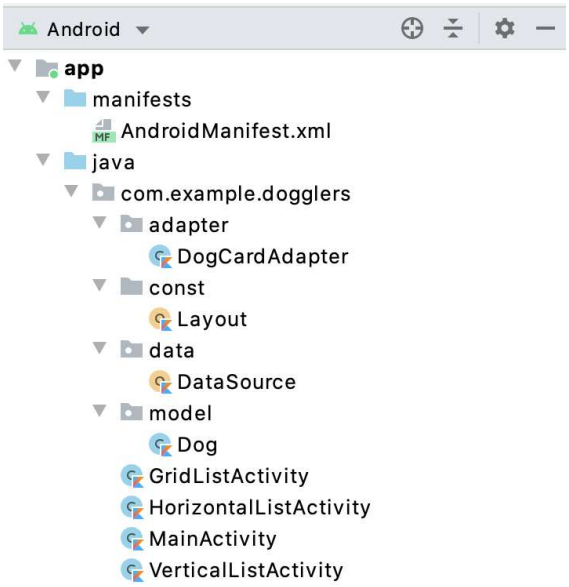


Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Otwórz** .



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.

Projekt podzielony jest na osobne pakiety. Chociaż większość funkcji jest już zaimplementowana, musisz zaimplementować `DogCardAdapter`. Istnieją również dwa pliki układu, które musisz zmodyfikować. Inne pliki są omówione w razie potrzeby w poniższych instrukcjach.



Zaimplementuj układ

Zarówno układ pionowy, jak i poziomy są identyczne, więc wystarczy zaimplementować jeden plik układu dla obu. Układ siatki wyświetla wszystkie te same informacje, ale imię psa, wiek i hobby są ułożone pionowo, więc w tym przypadku będziesz potrzebować osobnego układu. Oba układy wymagają czterech różnych widoków, aby wyświetlić informacje o każdym psie.

1. A `ImageView`ze zdjęciem psa
2. A `TextView`z imieniem psa
3. A `TextView`z wiekiem psa
4. A `TextView`z hobby psa

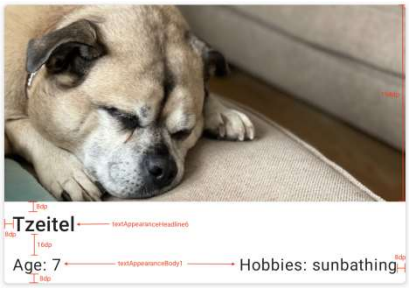
Zauważysz również stylizację na każdej karcie, aby pokazać obramowanie i cień. Jest to obsługiwane przez `MaterialCardView`program , który jest już dodany do plików układu w projekcie startowym. W każdym `MaterialCardView`jest miejsce , w `ConstraintLayout`którym musisz dodać pozostałe widoki.

Wskazówka: możesz użyć dowolnego identyfikatora dla każdego widoku, ale pamiętaj, że oba te układy będą używać tej samej `ViewHolder`klasy, więc upewnij się, że odpowiednie widoki w każdym układzie używają tego samego identyfikatora. Na przykład zarówno układ siatki, jak i układ poziomy/pionowy będą miały a `TextView`z id `dog_name`.

Istnieją dwa pliki XML, z którymi musisz pracować, aby zaimplementować układy: `vertical_horizontal_list_item.xml`dla układu poziomego i pionowego oraz `grid_list_item.xml`dla układu siatki.

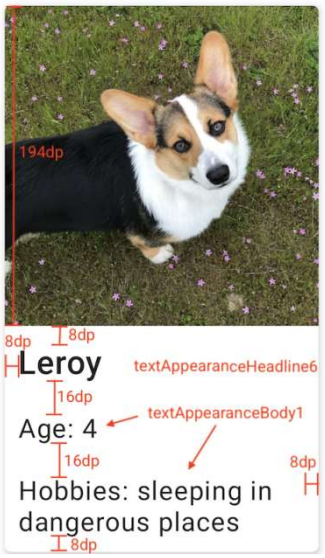
1. Zbuduj układ dla list pionowych i poziomych.

Otwórz `vertical_horizontal_list_item.xml`, a wewnątrz `ConstraintLayout`stwórz układ pasujący do obrazu.



2. Zbuduj układ siatki.

   Otwórz grid_list_item.xml, a wewnątrz ConstraintLayoutstwórz układ pasujący do obrazu.



Zaimplementuj adapter

Po zdefiniowaniu układów następnym zadaniem jest zaimplementowanie RecyclerViewadaptera. Otwórz DogCardAdapter.ktw pakiecie **adaptera .** Zobaczysz, że jest wiele TODOkomentarzy, które pomagają wyjaśnić, co musisz zaimplementować.

```
class DogCardAdapter(
    private val context: Context?,
    private val layout: Int
): RecyclerView.Adapter<DogCardAdapter.DogCardViewHolder>() {

    // TODO: Initialize the data using the List found in data/DataSource

    /**
     * Initialize view elements
     */
    class DogCardViewHolder(view: View?): RecyclerView.ViewHolder(view!!) {
        // TODO: Declare and initialize all of the list item UI components
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): DogCardViewHolder {
        // TODO: Use a conditional to determine the layout type and set it accordingly.
        //  if the layout variable is Layout.GRID the grid list item should be used. Otherwise the
        //  the vertical/horizontal list item should be used.

        // TODO Inflate the layout
```

Aby zaimplementować adapter, musisz wykonać pięć kroków.

1. Zdefiniuj zmienną lub stałą dla listy danych psa. Lista znajduje się w pakiecie **danych** w obiekcie o nazwie `DataSource`i wygląda następująco:

```
object DataSource {

  val dogs: List<Dog> = listOf( ...

}
```

Właściwość `dogs`jest typu `List<Dog>`. Klasa `Dog`znajduje się w pakiecie **modelu** i definiuje cztery właściwości: obraz (reprezentowany przez identyfikator zasobu) i trzy `String`właściwości.

```
data class Dog(
  @DrawableRes val imageResourceId: Int,
  val name: String,
  val age: String,
  val hobbies: String
)
```

Ustaw zmienną, którą definiujesz `DogCardAdapter`, na `dogs`listę w `DataSource`obiekcie.

2. Zaimplementuj `DogCardViewHolder`. Uchwyt widoku powinien powiązać cztery widoki, które należy ustawić dla każdej karty widoku recyklera. Ten sam uchwyt widoku będzie współdzielony dla obu układów `grid_list_item`i `vertical_horizontal_list_item`, ponieważ wszystkie widoki są współdzielone przez oba układy. `DogCardViewHolder`Powinny zawierać właściwości dla następujących identyfikatorów widoków : `dog_image`, `dog_name`, `dog_age`i `dog_hobbies`.
3. W `onCreateViewHolder()`programie chcesz zawyżyć układ `grid_list_item`lub `vertical_horizontal_list_item`. Skąd wiesz, którego układu użyć? W definicji adaptera widać, że `Int`podczas tworzenia instancji adaptera jest przekazywana wartość zwana układem typu.

```
class DogCardAdapter(
  private val context: Context?,
  private val layout: Int
): RecyclerView.Adapter<DogCardAdapter.DogCardViewHolder>() {
```

Odpowiada to wartości zdefiniowanej w `Layout`obiekcie znajdującym się w pakiecie **const .**

```
object Layout {
  val VERTICAL = 1
  val HORIZONTAL = 2
  val GRID = 3
}
```

Wartość układu będzie wynosić 1, 2 lub 3, ale można ją porównać z identyfikatorami `VERTICAL`, `HORIZONTAL`i `GRID`, z `Layout`obiektu.

W przypadku `GRID`układu powiększ `grid_list_item`układ, a w przypadku układów `HORIZONTAL`i — powiększ układ. Metoda powinna zwrócić instancję reprezentującą zawyżony układ.`VERTICAL`vertical_horizontal_list_itemDogCardViewHolder

Porada: Możesz użyć instrukcji warunkowej, takiej jak if lub when, aby ustawić zmienną na podstawie typu układu
(`grid_list_item`for `GRID`i `vertical_horizontal_list_item`for `VERTICAL`lub `HORIZONTAL`). Po uzyskaniu prawidłowego identyfikatora układu po prostu przekaż go do metody, aby napełnić układ.

4. Implementuj `getItemCount()`, aby zwrócić długość listy psów.
5. Na koniec musisz zaimplementować `onBindViewHolder()`ustawienie danych w każdej z kart widoku recyklera. Użyj , `position`aby uzyskać dostęp do poprawnych danych psa z listy i ustawić obraz i imię psa. Użyj zasobów tekstowych `dog_age`i , `dog_hobbies`aby odpowiednio sformatować wiek i hobby.

Porada: W `onBindViewHolder()`metodzie zdefiniowaliśmy `resources`zmienną, której można używać do odwoływania się do zasobów ciągów bez użycia za `context?.resources`każdym razem.

Po zakończeniu implementacji adaptera uruchom aplikację w emulatorze, aby sprawdzić, czy wszystko zostało poprawnie zaimplementowane.

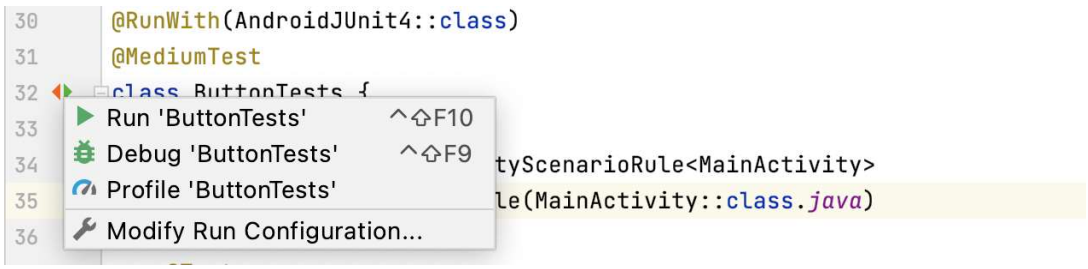
4. Przetestuj swoją aplikację

Projekt Dogglers zawiera cel „androidTest” z kilkoma przypadkami testowymi.

Przeprowadzanie testów

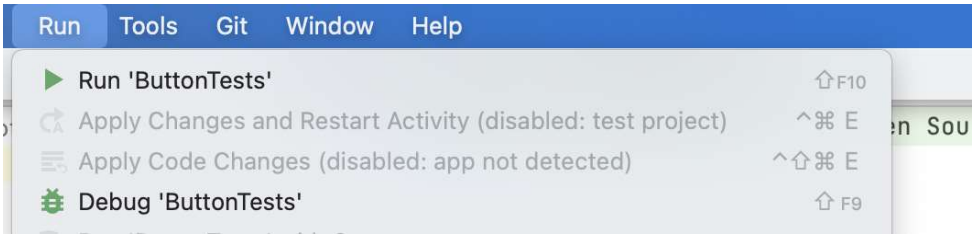
Aby uruchomić testy, możesz wykonać jedną z następujących czynności:

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.

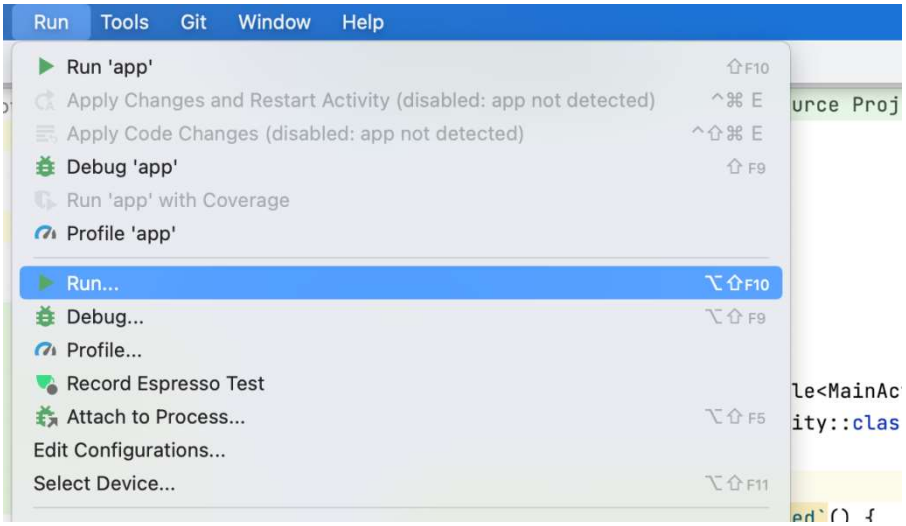


Często będziesz chciał uruchomić tylko jeden test, na przykład, jeśli tylko jeden test się nie powiedzie, a pozostałe testy zakończą się pomyślnie. Pojedynczy test można uruchomić tak samo, jak cały przypadek testowy. Użyj zielonej strzałki i wybierz opcję **Uruchom** .

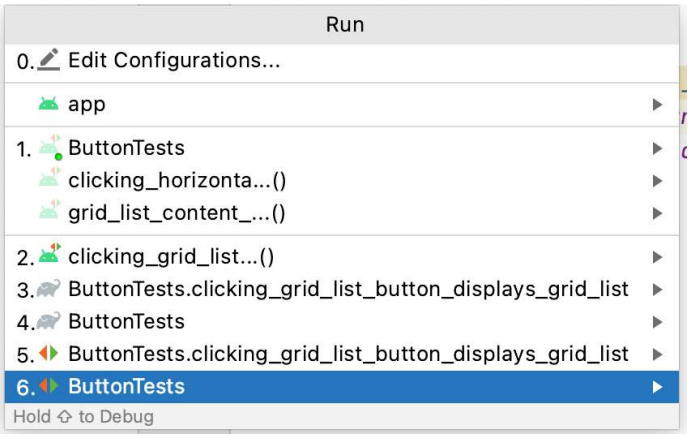
Jeśli masz wiele przypadków testowych, możesz również uruchomić cały zestaw testów. Podobnie jak w przypadku uruchamiania aplikacji, tę opcję znajdziesz w menu **Uruchom** .



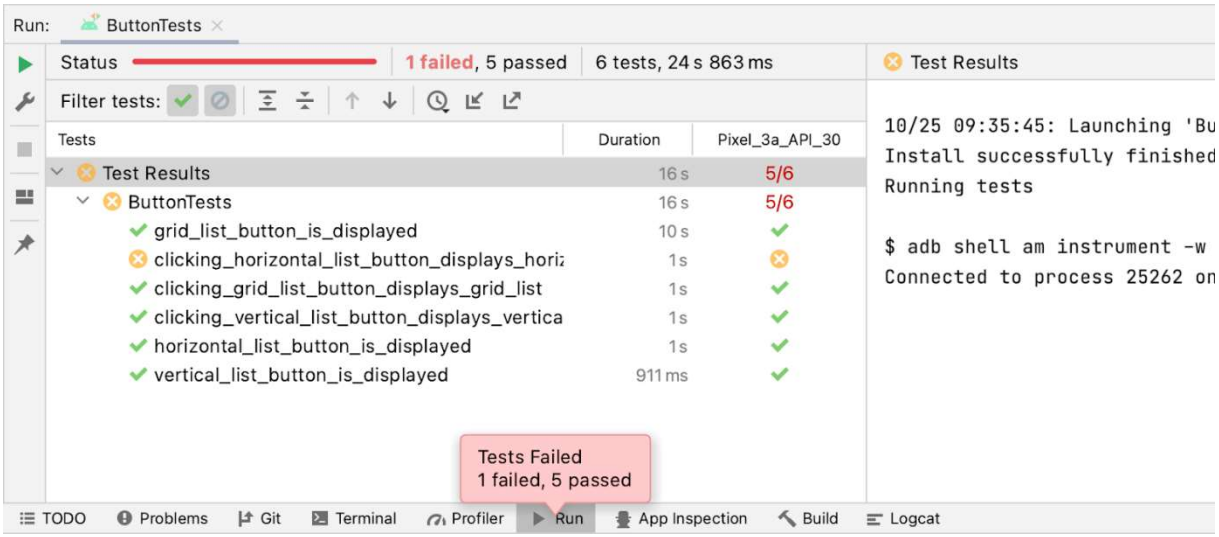
Zwróć uwagę, że Android Studio domyślnie dopasuje się do ostatniego uruchomionego celu (aplikacji, celów testowych itp.), więc jeśli w menu nadal pojawi się komunikat **Uruchom > Uruchom „aplikację”** , możesz uruchomić cel testu, wybierając **Uruchom > Uruchom** .



Następnie wybierz cel testowy z menu podręcznego.

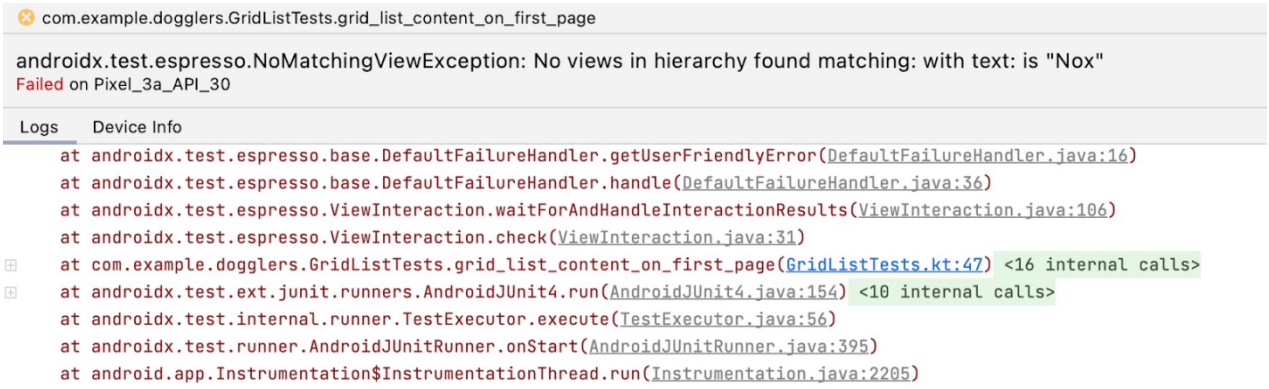


Wyniki uruchomienia testów są pokazane w zakładce **Uruchom .** W okienku po lewej stronie zobaczysz listę testów zakończonych niepowodzeniem, jeśli takie istnieją. Testy zakończone niepowodzeniem są oznaczone czerwonym wykrzyknikiem obok nazwy funkcji. Zaliczone testy są oznaczone zielonym haczykiem.



Porada: Aby wyświetlić testy zaliczone, niezaliczone lub zarówno zaliczone, jak i niezaliczone w lewym okienku, możesz użyć dwóch przycisków w lewym górnym rogu. Zaznaczenie znacznika wyboru pokaże zaliczone testy, wybranie ikony okręgu z linią przez nią wyświetli listę nieudanych testów. Domyślnie wyświetlane są tylko testy zakończone niepowodzeniem.

Jeśli test się nie powiedzie, dane wyjściowe tekstowe zawierają informacje pomocne w rozwiązaniu problemu, który spowodował niepowodzenie testu.



Na przykład w powyższym komunikacie o błędzie test sprawdza, czy wyświetlany jest ciąg zawierający słowo „Nox”. Jednak test kończy się niepowodzeniem. Nie można znaleźć tekstu, co oznacza, że prawdopodobnie nie jest jeszcze wyświetlany.