Część 1: Podstawy Kotlina

Twórz swoje pierwsze aplikacje na Androida za pomocą języka programowania Kotlin. Dodaj obrazy i tekst do aplikacji na Androida oraz dowiedz się, jak używać klas, obiektów i warunków warunkowych do tworzenia interaktywnej aplikacji dla użytkowników.

Spis treści

Ścieżka 1. Wprowadzenie do Kotlina

Poznaj podstawy Kotlina, nowoczesnego języka programowania, który pozwala w zwięzły sposób wyrażać swoje pomysły.

Napisz swój pierwszy program w Kotlin

1. Zanim zaczniesz

W tym laboratorium kodowania napiszesz swój pierwszy program w języku Kotlin, korzystając z interaktywnego edytora, który możesz uruchomić z przeglądarki.

Możesz myśleć o *programie* jako o serii instrukcji dla systemu, aby wykonać jakąś akcję. Na przykład możesz napisać program, który utworzy kartkę urodzinową. W tym programie można było napisać instrukcję, aby wydrukować tekst gratulacyjny lub obliczyć czyjś wiek od roku urodzenia.

Tak jak używasz ludzkiego języka do komunikowania się z drugą osobą, używasz języka programowania do komunikowania się z systemem operacyjnym swojego komputera. Na szczęście języki programowania są mniej złożone niż języki ludzkie i całkiem logiczne!

Aplikacje na Androida są napisane w języku programowania Kotlin. Kotlin to nowoczesny język stworzony, aby pomóc programistom pisać kod efektywnie i z jak najmniejszą liczbą błędów.

Nauka tworzenia aplikacji i równoczesna nauka podstaw programowania będzie nie lada wyzwaniem, więc zaczniemy od odrobiny programowania, zanim przejdziemy do tworzenia aplikacji. Zaznajomienie się z pewnymi podstawami programowania jest nie tylko ważnym krokiem w kierunku tworzenia aplikacji, ale także ułatwi tworzenie pierwszej aplikacji w dalszej części tego kursu.

Edytory kodu to narzędzia, które pomagają pisać kod, w taki sam sposób, w jaki edytor tekstu (taki jak Dokumenty Google) pomaga tworzyć dokumenty tekstowe. W tym ćwiczeniu z kodowania używasz interaktywnego edytora Kotlin w swojej przeglądarce. Oznacza to, że nie musisz instalować żadnego oprogramowania, aby zrobić pierwszy krok w kierunku tworzenia aplikacji.

Warunki wstępne

- Korzystaj z interaktywnych stron internetowych w swojej przeglądarce internetowej.

Czego się nauczysz

- Jak stworzyć, zmienić, zrozumieć i uruchomić minimalny program Kotlin, który wyświetla komunikat.

Co zbudujesz

- Program w języku programowania Kotlin, który wyświetla komunikat po uruchomieniu.

Czego potrzebujesz

- Komputer z nowoczesną przeglądarką internetową, np. najnowszą wersją Chrome .
- Dostęp do Internetu na Twoim komputerze.

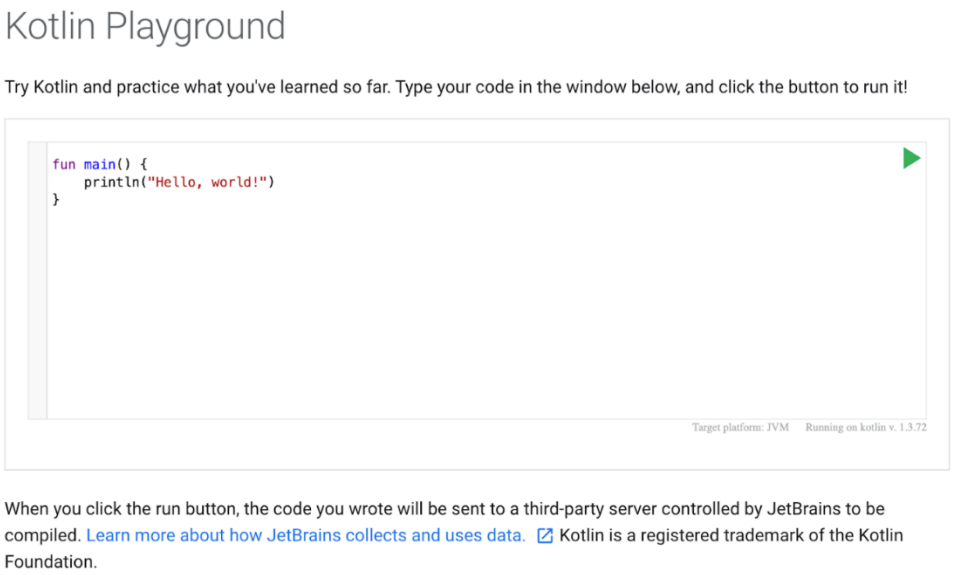
2. Uruchom swój pierwszy program w Kotlin

W tym zadaniu użyjesz edytora na stronie internetowej, aby od razu rozpocząć programowanie w języku Kotlin.

Użyj interaktywnego edytora kodu

Zamiast instalować oprogramowanie na komputerze, możesz użyć narzędzia internetowego do stworzenia pierwszego programu.

1. W przeglądarce otwórz https://developer.android.com/training/kotlinplayground . Spowoduje to otwarcie narzędzia programistycznego opartego na przeglądarce.
2. Powinieneś zobaczyć stronę podobną do poniższego zrzutu ekranu, z edytorem kodu pośrodku.



Oto kod programu w edytorze:

```
fun main() {
  println("Hello, world!")
```

}

Uruchom kod programu

Uruchamianie utworzonego programu niewiele różni się od uruchamiania programu, takiego jak edytor tekstu na komputerze. Różnica polega na tym, że kiedy uruchamiasz program w celu wykonania zadania lub grania w grę, zależy ci przede wszystkim na tym, co program może dla ciebie zrobić, a nie zajmujesz się kodem, który sprawia, że działa. Kiedy programujesz, możesz zobaczyć i pracować z rzeczywistym kodem, który sprawia, że dzieje się magia.

Zobaczmy, co robi ten program!

1. W edytorze w prawym górnym rogu znajdź biały lub zielony trójkąt  i kliknij go, aby uruchomić program.
2. Spójrz na okienko na dole.

```
Hello, world!
```

1. Uwaga `Hello, world!`wydrukowana, jak na powyższym obrazku. Więc teraz wiesz, co robi ten program: drukuje lub wyświetla komunikat „Witaj świecie”.

Kompilacja to proces, który tłumaczy kod programu Kotlin na postać, którą system może uruchomić. Jeśli kompilacja zakończy się pomyślnie, w programie nie ma błędów, które uniemożliwiłyby jego uruchomienie. Jeśli pojawią się problemy, pojawią się w okienku na dole.

3. Zmodyfikuj swój program

Zmień kod Hello World

Zmieńmy program, aby robił coś trochę innego.

1. Zmień `"Hello, world!"`tekst na `"Happy Birthday!"`.
2. Uruchom swój program, klikając niebieski lub zielony przycisk uruchamiania w prawym górnym rogu.
3. Na dole powinieneś zobaczyć `Happy Birthday!`wydruk, jak pokazano poniżej.

```
Happy Birthday!
```

Jak to działa?

Jak to się robi? Wydaje się, że to dużo kodu, żeby coś wydrukować!

Cóż, jeśli chciałbyś, aby znajomy napisał "Hello, world!" na kartce papieru jest wiele ukrytych informacji. Jeśli po prostu im powiesz: „Napisz 'Witaj świecie!' na tej kartce papieru”, zrobią założenia dotyczące informacji, które pominąłeś. Na przykład założą, że muszą użyć długopisu i że chcesz, aby napisali to literami! Komputer nie przyjmuje tych założeń, więc musisz podać dokładne instrukcje, które obejmują każdy krok.

Tak jak język angielski ma strukturę, tak samo ma język programowania. Jeśli kiedykolwiek uczyłeś się innego języka, znasz wyzwanie związane z nauką gramatyki, pisowni, być może nowego alfabetu symboli i słownictwa. Nauka programowania wiąże się z podobnymi wyzwaniami, ale na szczęście jest mniej skomplikowana i dużo bardziej logiczna niż nauka np. angielskiego.

Zrozum części programu

Teraz spójrz na kod. Każdy element tego programu służy określonemu celowi i potrzebujesz wszystkich elementów, aby móc uruchomić program. Zacznijmy od pierwszego słowa.

fun

- `fun`to słowo w języku programowania Kotlin. `fun`oznacza funkcję. Funkcja to sekcja programu, która wykonuje określone zadanie.

Uwaga: Kotlin ma wiele specjalnych słów o bardzo konkretnych znaczeniach. Gdy nauczysz się programować w języku Kotlin, nauczysz się tych słów. Często nazywa się je słowami kluczowymi lub słowami zastrzeżonymi.

fun main

- `main`to nazwa tej funkcji. Funkcje mają nazwy, dzięki czemu można je od siebie odróżnić. Ta funkcja nosi nazwę `main`, ponieważ jest pierwszą lub główną funkcją, która jest wywoływana po uruchomieniu programu. Każdy program Kotlina potrzebuje funkcji o nazwie `main`.

fun main()

- Po nazwie funkcji zawsze występują `()`dwa nawiasy.
- Wewnątrz nawiasów możesz umieścić informacje, których funkcja ma używać. To wejście do funkcji nazywa się "argumentami" lub `args`w skrócie. Więcej o kłótniach dowiesz się później.

fun main() {}

- Zwróć uwagę na parę nawiasów klamrowych `{}`po nawiasach. Wewnątrz funkcji znajduje się kod, który wykonuje zadanie. Te nawiasy klamrowe otaczają te wiersze kodu.

Spójrz na linię kodu między nawiasami klamrowymi:

println("Happy Birthday!")

Ten wiersz kodu drukuje `Happy Birthday!` tekst.

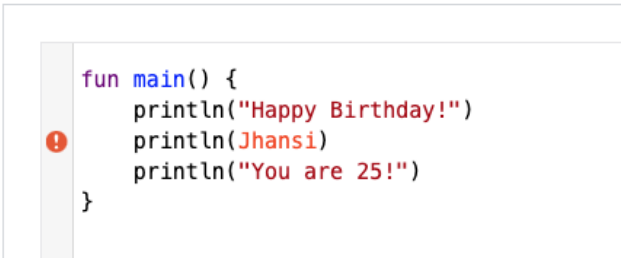
- `println`nakazuje systemowi wydrukowanie wiersza tekstu.
- Wewnątrz nawiasów umieszczasz tekst do wydrukowania.
- Zauważ, że tekst do wydrukowania jest otoczony cudzysłowami. To mówi systemowi, że wszystko w cudzysłowie powinno być wydrukowane dokładnie tak, jak podano.

Aby faktycznie wydrukować tekst, cała ta `println`instrukcja musi znajdować się wewnątrz `main`funkcji.

Tak więc jest. Najmniejszy program Kotlin.

```
fun main() {
  println("Happy Birthday!")
}
```

4. Rozszerz swój program



Wydrukuj więcej niż jedną wiadomość

Dobra robota! Wydrukowałeś jeden wiersz tekstu za pomocą `println() function`. Możesz jednak umieścić w funkcji tyle linii instrukcji, ile chcesz lub potrzebujesz, aby wykonać zadanie.

1. Skopiuj linię `println("Happy Birthday!")`i wklej ją jeszcze dwa razy pod nią. Upewnij się, że wklejone linie znajdują się w nawiasach klamrowych `main`funkcji.
2. Zmień jeden tekst do wydrukowania na czyjeś imię, powiedz „Jhansi”.
3. Zmień inny tekst do wydrukowania na „Masz 25 lat!”.

Twój kod powinien wyglądać jak poniższy kod.

```
fun main() {
  println("Happy Birthday!")
  println("Jhansi")
  println("You are 25!")
}
```

Czego można oczekiwać od tego kodu po uruchomieniu?

1. Uruchom swój program, aby zobaczyć, co robi.
2. Przejdź do okienka wyjściowego i powinieneś zobaczyć 3 linie wydrukowane w oknie konsoli, jak pokazano poniżej.

```
Happy Birthday!
Jhansi
You are 25!
```

Dobra robota!

Radzenie sobie z błędami

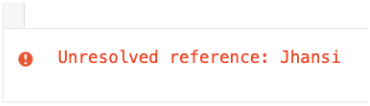
Popełnianie błędów podczas programowania jest normalne, a większość narzędzi zapewnia informacje zwrotne, które pomagają naprawić błędy. Na tym etapie stwórz błąd, aby zobaczyć, co się stanie.

1. W swoim programie usuń cudzysłowy wokół tekstu `Jhansi`, aby linia wyglądała tak, jak pokazano poniżej.

```
println(Jhansi)
```

1. Uruchom swój program. Powinieneś zobaczyć `Jhansi`wydrukowany na czerwono i wykrzyknik obok zmienionego wiersza kodu, aby pokazać, gdzie wystąpił błąd.

2. Spójrz na okienko danych wyjściowych. Pokazuje wiadomość z tą samą ikoną wykrzyknika i słowem `Error`. Poniżej znajduje się opis błędu w twoim kodzie.



1. Ten komunikat, `Unresolved reference: Jhansi`, informuje, co według systemu jest błędem w kodzie. Nawet jeśli nie wiesz, co oznacza komunikat o błędzie, możesz dowiedzieć się, co jest nie tak. W takim przypadku wiesz, że `println()`instrukcja drukuje tekst. Dowiedziałeś się wcześniej, że tekst musi znajdować się między cytatami. Jeśli tekst nie jest cytowany, jest to błąd.
2. Śmiało i ponownie dodaj cytaty.
3. Uruchom swój program, aby upewnić się, że znowu działa.

Gratulacje, uruchomiłeś i zmieniłeś swój pierwszy program Kotlin!

5. Kod rozwiązania

To jest kompletny kod programu, nad którym pracowałeś w tym laboratorium.

```
fun main() {
   println("Happy Birthday!")
   println("Jhansi")
   println("You are 25!")
}
```

6. Podsumowanie

- https://developer.android.com/training/kotlinplayground to interaktywny edytor kodu w sieci, w którym możesz ćwiczyć pisanie programów Kotlin.
- Wszystkie programy Kotlin muszą mieć `main()`funkcję:`fun main() {}`
- Użyj `println()`funkcji, aby wydrukować wiersz tekstu.
- Umieść tekst, który chcesz wydrukować, między podwójnymi cudzysłowami. Na przykład `"Hello"`.
- Powtórz `println()`instrukcję, aby wydrukować wiele wierszy tekstu.
- Błędy są w programie zaznaczone na czerwono. W okienku danych wyjściowych znajduje się komunikat o błędzie, który pomaga ustalić, gdzie jest błąd i co może go powodować.

7. Dowiedz się więcej

- https://developer.android.com/training/kotlinplayground
- Słownictwo dla Androida Podstawy w Kotlin

8. Ćwicz na własną rękę

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

Wykonaj następujące czynności:

1. Zmień `println()`instrukcje na `print()`.
2. Uruchom swój program. Co się dzieje?

Wskazówka: Instrukcja `print()`drukuje tylko tekst bez dodawania łamania wiersza na końcu każdego ciągu.

1. Popraw tekst tak, aby każda część wiadomości znajdowała się w osobnym wierszu.

Wskazówka: użyj `\n`wewnątrz tekstu, aby dodać podział wiersza. Na przykład `"line \n break"`. Dodanie łamania wiersza zmienia wynik, jak pokazano poniżej.

Podpowiedź: Możesz wydrukować pustą linię, nie podając tekstu: `println("")`.

Kod:

```
fun main() {
   println("no line break")
   println("")
   println("with line \n break")
}
```

Wyjście:

```
no line break

with line
 break
```

Sprawdź swoją pracę:

Oto jedno z możliwych rozwiązań:

```
fun main() {
   print("Happy Birthday!\n")
   print("Jhansi\n")
   print("You are 25!")
}
```

Utwórz wiadomość urodzinową w Kotlin

1. Wstęp

W tym ćwiczeniu z kodowania utworzysz krótki program Kotlin, który drukuje wiadomość urodzinową z tortem i banerem urodzinowym.

Warunki wstępne

- Jak otwierać i edytować kod w https://developer.android.com/training/kotlinplayground , przeglądarkowym narzędziu do programowania Kotlin.
- Zapoznaj `"Hello world!"`się z programem z kursu „Napisz swój pierwszy program z kodem Kotlin”.
- Jak używać `println()`do pisania tekstu do konsoli internetowego edytora kodu Kotlin.

Czego się nauczysz

- Jak wydrukować bardziej złożony tekst z Twojego programu.
- Jak wykonać podstawową matematykę w Kotlinie i przechowywać wyniki w zmiennych do późniejszego wykorzystania.
- Jak utworzyć funkcję, która wypisze ten sam ciąg kilka razy.
- Jak utworzyć pętlę, która wielokrotnie drukuje fragment tekstu.

Co zbudujesz

- Stworzysz krótki program, za pomocą którego możesz drukować wiadomości urodzinowe, tekstowe zdjęcie tortu i baner.

Czego potrzebujesz

- Komputer z dostępem do internetu i nowoczesną przeglądarką internetową, np. najnowszą wersją Chrome .

2. Utwórz wiadomość urodzinową w Kotlin

Skonfiguruj swój kod startowy

1. W przeglądarce otwórz https://developer.android.com/training/kotlinplayground . Spowoduje to otwarcie narzędzia do programowania Kotlin opartego na przeglądarce.
2. Wewnątrz `fun main()`funkcji zamień `"Hello, world!"`tekst na `"Happy Birthday, Rover!"`.
3. Poniżej, nadal w nawiasach klamrowych, dodaj jeszcze dwie linie do wydrukowania: `"You are already 5!"`i`"5 is the very best age to celebrate!"`.

Twój gotowy kod powinien wyglądać tak.

```
fun main() {
  println("Happy Birthday, Rover!")
  println("You are already 5!")
```

```
    println("5 is the very best age to celebrate!")
}
```

1. Uruchom swój kod.
2. Sprawdź, czy okienko danych wyjściowych pokazuje **Wszystkiego najlepszego, Rover!** a poniżej **masz już 5 lat!** a **5 to najlepszy wiek do świętowania!**

```
Happy Birthday, Rover!
You are already 5!
5 is the very best age to celebrate!
```

Dodaj tort urodzinowy

Wiadomość urodzinowa wymaga zdjęcia o tematyce urodzinowej. Jak ciasto. Możesz dodać tort do wiadomości urodzinowej, drukując dodatkowe wiersze zawierające litery i symbole na klawiaturze oraz `println()`.

Kontynuuj od powyższego kodu rozwiązania.

1. W kodzie między dwiema `println()`instrukcjami for `Happy Birthday`i `You are already 5`dodaj następujące wiersze instrukcji print, jak pokazano poniżej. To tworzy ciasto. Ostatnia `println()`instrukcja nie zawiera tekstu między cudzysłowami, co powoduje wypisanie pustej linii.

```
    println("   ,,,,,   ")
    println("   |||||   ")
    println(" =========")
    println("@@@@@@@@@@@@")
    println("{~@~@~@~@~@~}")
    println("@@@@@@@@@@@@")
    println("")
```

Aby pomóc innym zrozumieć Twój kod, możesz dodać komentarz przed wydrukowaniem ciasta. Jeśli uruchomisz swój kod, wynik nie będzie wyglądał inaczej, ponieważ komentarze to tylko informacje dla Ciebie i innych programistów, a nie polecenia dla systemu. Komentarz w wierszu zaczyna się `//`od tekstu, po którym następuje tekst, jak pokazano poniżej.

```
// This is a comment line
// This is another comment
```

1. Dodaj komentarz przed wydrukowaniem ciasta: `// Let's print a cake!`.
2. Dodaj komentarz przed wydrukowaniem pustej linii: `// This prints an empty line`.

Twój kod powinien wyglądać jak poniższy kod.

```
fun main() {
    println("Happy Birthday, Rover!")

    // Let's print a cake!
    println("   ,,,,,   ")
```

```
    println("   |||||   ")
    println(" =========")
    println("@@@@@@@@@@")
    println("{~@~@~@~@~}")
    println("@@@@@@@@@@")

    // This prints an empty line.
    println("")

    println("You are already 5!")
    println("5 is the very best age to celebrate!")
}
```

Wskazówka: Zwróć uwagę, że dodaliśmy trochę spacji (pustych linii) w kodzie, aby oddzielić sekcje kodu. Dzięki temu kod jest bardziej czytelny. Możesz dodać puste wiersze wszędzie tam, gdzie uznasz to za przydatne.

1. Uruchom swój kod, a wynik powinien wyglądać jak poniżej.

```
Happy Birthday, Rover!
  ,,,,,
   |||||
 =========
@@@@@@@@@@
{~@~@~@~@~}
@@@@@@@@@@

You are already 5!
5 is the very best age to celebrate!
```

3. Twórz i używaj zmiennych

Przechowuj wiek łazika w zmiennej

1. W gotowym kodzie do tej pory zwróć uwagę, jak dwukrotnie powtarzasz ten sam numer wieku.

Zamiast powtarzać tę liczbę, możesz przechowywać ją w jednym miejscu jako zmienną. To tak, jakbyś włożył swój numer do pudełka i nadał mu imię. Następnie możesz użyć tej nazwy zmiennej za każdym razem, gdy potrzebujesz wartości. A jeśli zmieni się wiek, wystarczy zmienić program w jednym miejscu. Zmieniając zmienną, poprawna wartość wieku jest drukowana wszędzie tam, gdzie używana jest zmienna.

1. W swoim programie, jako pierwszy wiersz kodu wewnątrz `main()`funkcji, dodaj następujący kod, aby utworzyć zmienną o nazwie `age`, o wartości 5, jak pokazano poniżej. (Musisz umieścić tę linię przed `println()`oświadczeniami).

```
val age = 5
```

Ta linia oznacza:

- `val`to specjalne słowo używane przez Kotlina, zwane *słowem kluczowym* , wskazujące, że następuje nazwa zmiennej.
- `age`to nazwa zmiennej.
- `=`sprawia, że wartość `age`(po jego lewej stronie) będzie taka sama jak wartość po jego prawej stronie. W matematyce pojedynczy znak równości służy do stwierdzenia, że wartości po obu stronach są takie same. W Kotlinie, w przeciwieństwie do matematyki, pojedynczy znak równości służy do przypisania wartości po prawej stronie do nazwanej zmiennej po lewej stronie.

Deweloper powiedziałby to tak: Ta linia deklaruje zmienną o nazwie, `age`której przypisana wartość to `5`.

Ważne: Zmienną zadeklarowaną przy użyciu `val`słowa kluczowego można ustawić tylko raz. Nie możesz później zmienić jego wartości w programie.

Możesz zadeklarować zmienną zmienną za pomocą `var`słowa kluczowego, co zrobisz w późniejszym ćwiczeniu z programowania.

Aby użyć zmiennej wewnątrz instrukcji print, musisz otoczyć ją pewnymi symbolami, które informują system, że to, co nastąpi dalej, nie jest tekstem, ale zmienną. Zamiast drukować tekst, system musi wydrukować wartość zmiennej. Robisz to, umieszczając zmienną w nawiasach klamrowych poprzedzonych znakiem dolara, jak w poniższym przykładzie.

```
${variable}
```

1. W swoim kodzie zastąp liczbę 5 w obu instrukcjach print `age`zmienną, jak pokazano poniżej.

```
println("You are already ${age}!")
println("${age} is the very best age to celebrate!")
```

1. Uruchom swój kod, a obie wiadomości powinny pokazywać ten sam wiek.
2. Zmień wartość zmiennej na inną. Na przykład możesz pokazać wiek łazika w dniach zamiast w latach. Aby to zrobić, pomnóż wiek przez 365, pomijając lata przestępne. Możesz wykonać to obliczenie bezpośrednio podczas tworzenia zmiennej, jak pokazano poniżej.

```
val age = 5 * 365
```

1. Uruchom kod ponownie i zauważ, że obie wiadomości pokazują teraz wiek w dniach.

```
Happy Birthday, Rover!
  ,,,,,
  |||||
 =========
@@@@@@@@@@@
{~@~@~@~@~}
@@@@@@@@@@@

You are already 1825!
1825 is the very best age to celebrate!
```

1. **[Opcjonalnie]** Zmień tekst komunikatów drukowanych, aby działał lepiej z dniami. Na przykład zmień je na:

```
You are already 1825 days old!
1825 days old is the very best age to celebrate!
```

Umieść tekst w zmiennej

Do zmiennych można wstawiać nie tylko liczby, ale także tekst.

1. Poniżej zmiennej dla `age`dodaj zmienną nazwaną `name`dla imienia osoby, która urodziła się i ustaw jej wartość na `"Rover"`.

```
val name = "Rover"
```

1. Zastąp imię i nazwisko `Rover`w wiadomości o urodzinach zmienną, jak pokazano poniżej.

```
println("Happy Birthday, ${name}!")
```

I możesz mieć więcej niż jedną zmienną w instrukcji print.

1. Dodaj `Rover`do wiadomości o wieku, używając `name`zmiennej, jak pokazano poniżej.

```
println("You are already ${age} days old, ${name}!")
```

Twój ukończony kod powinien wyglądać podobnie do tego.

```
fun main() {

    val age = 5 * 365
    val name = "Rover"

    println("Happy Birthday, ${name}!")

    // Let's print a cake!
    println("   ,,,,,   ")
    println("   |||||   ")
    println(" =========")
    println("@@@@@@@@@@@")
    println("{~@~@~@~@~}")
    println("@@@@@@@@@@@")

    // This prints an empty line.
    println("")

    println("You are already ${age} days old, ${name}!")
    println("${age} days old is the very best age to celebrate!")
}
```

Gratulacje! Możesz teraz tworzyć wiadomości z tekstem, grafiką utworzoną z symboli, używać zmiennych do przechowywania liczb i tekstu oraz drukować tekst za pomocą zmiennych.

4. Wydrukuj baner urodzinowy z ramką

W tym zadaniu utworzysz baner urodzinowy, a następnie nauczysz się uprościć ten kod za pomocą technik powtarzania i ponownego użycia kodu oraz dlaczego to dobrze.

Utwórz baner urodzinowy dla początkujących

1. Na https://developer.android.com/training/kotlinplayground umieść kursor gdzieś wewnątrz kodu.
2. Kliknij prawym przyciskiem myszy, aby otworzyć menu i wybierz **opcję Zaznacz wszystko** .
3. Naciśnij klawisz Backspace lub Delete, aby usunąć cały kod.
4. Skopiuj i wklej poniższy kod do edytora.

```
fun main() {
  println("========================")
  println("Happy Birthday, Jhansi!")
  println("========================")
}
```

1. Uruchom swój program, aby zobaczyć baner wydrukowany w konsoli.

```
========================
Happy Birthday, Jhansi!
========================
```

Utwórz funkcję do drukowania obramowania

Kod, który właśnie wkleiłeś i uruchomiłeś, to wywołana funkcja `main()`, która zawiera trzy instrukcje print. Po naciśnięciu przycisku **Uruchom** system wykonuje funkcję i cały zawarty w niej kod.

Podsumowanie

Podczas poprzedniego ćwiczenia z programowania dowiedziałeś się, że:

- Funkcja to sekcja programu, która wykonuje określone zadanie.
- Słowo `fun`kluczowe oznacza pewien kod jako funkcję.
- Po `fun`słowie kluczowym następuje nazwa funkcji, nawiasy dla opcjonalnych danych wejściowych funkcji (argumenty) i nawiasy klamrowe.
- Twój kod do drukowania tekstu zawsze znajdował się w tych nawiasach klamrowych.

Twój program Kotlin zawsze musi mieć `main()`funkcję. Ponadto możesz tworzyć i korzystać z własnych funkcji. Podobnie jak zmienne pomagają uniknąć duplikowania pracy, funkcje mogą pomóc uniknąć wielokrotnego pisania tego samego kodu. W twoim kodzie instrukcje print na górze i na dole banera są dokładnie takie same. Stwórzmy i użyjmy funkcji do drukowania tych obramowań.

1. W edytorze pod `main()`funkcją wstawiamy pustą linię, żeby mieć trochę miejsca do pracy. System ignoruje puste wiersze i możesz je wstawiać tam, gdzie są pomocne przy porządkowaniu kodu.
2. Utwórz funkcję. Zacznij od `fun`słowa kluczowego, po którym następuje nazwa , `printBorder`para nawiasów `()`i para nawiasów klamrowych `{}`, jak pokazano poniżej.

```
fun printBorder() {}
```

Słowo o nazywaniu funkcji.

- Zwróć uwagę, że nazwa funkcji `printBorder`zaczyna się od małej litery i czasownika. Nazwy funkcji prawie zawsze zaczynają się od małej litery i czasownika, a nazwa powinna opisywać działanie funkcji. Jak: `print()`lub tutaj, `printBorder()`.
- Zauważ również, że drugie słowo w nazwie zaczyna się od wielkiej litery. Ten styl nazywa się „skrzynią wielbłąda" i sprawia, że nazwy są znacznie łatwiejsze do odczytania. Więcej przykładów nazw to `drawReallyCoolFancyBorder`i `printBirthdayMessage`.

Uwaga: Nazywanie takich funkcji to „konwencja kodowania", porozumienie między programistami dotyczące formatowania kodu. Formatowanie całego kodu w podobny sposób ułatwia czytanie i uczenie się na podstawie kodu napisanego przez innych programistów. Gdy zobaczysz kod od innych programistów Androida, zwykle będzie on sformatowany przy użyciu tych konwencji.

Aby dowiedzieć się więcej o kodowaniu formatowania, wszystkie konwencje znajdziesz w oficjalnym przewodniku po stylach pod [adresem https://developer.android.com/kotlin/style-guide](https://developer.android.com/kotlin/style-guide) . W tym przewodniku jest wiele, ale jeśli jesteś ciekawy, zajrzyj.

1. Umieść zamykający nawias klamrowy `}`funkcji `printBorder`w nowej linii i dodaj pustą linię między dwoma nawiasami klamrowymi, aby mieć miejsce na dodanie większej ilości kodu. Posiadanie nawiasu zamykającego `}`na własnej linii ułatwia zobaczenie, gdzie kończy się funkcja.
2. Wewnątrz `main()`funkcji skopiuj instrukcję print dla granicy i wklej ją między nawiasami klamrowymi `printBorder()`funkcji.

Twoja ukończona `printBorder()`funkcja powinna wyglądać tak.

```
fun printBorder() {
  println("======================")
}
```

Aby użyć lub wywołać funkcję, użyj jej nazwy w nawiasach. Zauważ, że w ten sposób używasz `println()`! Aby użyć tej `printBorder`funkcji, wywołaj `printBorder()`w kodzie dowolne miejsce, które jest potrzebne.

1. W `main()`funkcji zastąp wiersze kodu, które drukują linię obramowania za `println()`pomocą wywołań `printBorder()`funkcji. Twój gotowy kod powinien wyglądać tak.

```
fun main() {
  printBorder()
  println("Happy Birthday, Jhansi!")
  printBorder()
}
```

```
fun printBorder() {
  println("========================")
}
```

1. Uruchom swój kod, aby upewnić się, że wszystko działa jak poprzednio.

Zwróć uwagę, że zmiana kodu w celu usprawnienia lub ułatwienia pracy bez zmiany danych wyjściowych nazywana jest *„ refaktoryzacją ”*.

Powtórz wzór obramowania

Patrząc na linię graniczną, to naprawdę znowu ten sam symbol. Więc zamiast mówić:

„Wydrukuj ten ciąg 23 symboli”

można powiedzieć,

„Wydrukuj ten 1 symbol 23 razy”.

W kodzie robi się to za pomocą `repeat()`instrukcji.

1. W `printBorder()`funkcji użyj `repeat()`instrukcji, aby wydrukować znak równości 23 razy.
2. Zamiast używać `println()`, użyj `print()`, aby nie przeskakiwać do nowej linii po wydrukowaniu każdego "=".

Oto kod. Masz teraz pojedynczą instrukcję, aby wydrukować znak równości i aby powtórzyć tę instrukcję 23 razy, używasz `repeat()`instrukcji.

```
fun printBorder() {
  repeat(23) {
    print("=")
  }
}
```

- Wypowiedź `repeat()`zaczyna się od słowa `repeat`, po którym następuje `()`. Ten rodzaj instrukcji jest określany jako „pętla”, ponieważ wielokrotnie powtarzasz lub zapętlasz ten sam kod. O innych sposobach tworzenia pętli dowiesz się później.
- W nawiasach `()`znajduje się liczba powtórzeń,
- a następnie nawiasy klamrowe`{}`,
- a wewnątrz nawiasów klamrowych `{}`kod do powtórzenia.

1. W `printBorder()`funkcji, po zamykającym nawiasie klamrowym } `repeat()`instrukcji, czyli po zakończeniu drukowania linii obramowania, dodaj `println()`instrukcję, aby wydrukować znak nowej linii.

Twój kod powinien teraz wyglądać tak.

```
fun printBorder() {
  repeat(23) {
```

```
    print("=")
  }
  println()
}
```

Kod w `main()`funkcji nie ulega zmianie, a cały program powinien wyglądać tak.

```
fun main() {
  printBorder()
  println("Happy Birthday, Jhansi!")
  printBorder()
}

fun printBorder() {
  repeat(23) {
    print("=")
  }
  println()
}
```

1. Uruchom swój kod. Dane wyjściowe powinny być takie same jak poprzednio, ale tym razem można było utworzyć obramowanie, określając symbol „=” tylko raz!

```
=======================
Happy Birthday, Jhansi!
=======================
```

Użyj argumentów, aby zmienić granicę

Co by było, gdybyś chciał stworzyć obramowania, które używają różnych symboli, takich jak te poniżej?

```
%%%%%%%%%%%%%%%%%%%%%%

:::::::::::::::::::::::::::::::::::::::::: ::::::::::::::::::::::
```

Możesz zdefiniować oddzielną funkcję dla każdego z tych różnych symboli. Jest jednak na to skuteczniejszy sposób. Możesz ponownie użyć funkcji, którą już napisałeś i uczynić ją bardziej elastyczną, aby działała dla różnych rodzajów symboli.

Fajną rzeczą w funkcjach jest to, że możesz podać im dane wejściowe za pomocą argumentów. Zetknąłeś się z tym krótko podczas poprzedniego ćwiczenia z programowania, kiedy zapoznałeś się z `main()`. W tym kroku dodasz argument do `printBorder()`funkcji, aby mogła wydrukować dowolny wzór obramowania, który dostarczysz.

1. W `main()`, u góry, utwórz zmienną o nazwie `border`wzór obramowania. To zatrzyma tekst do powtórzenia dla obramowania.

```
val border = "%"
```

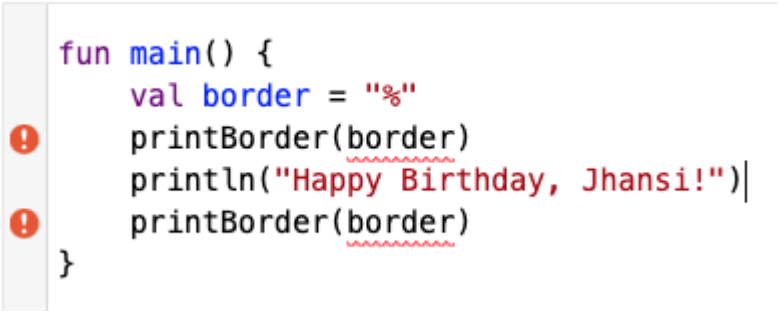
1. Teraz przekaż tę `border`zmienną do obu wywołań `printBorder()`funkcji jako argument. Robisz to, umieszczając `border`w nawiasach `()`, tak jak wtedy, gdy podałeś tekst `println()`do wydrukowania.

Twoja `main()`funkcja powinna wyglądać jak poniższy kod.

```
fun main() {
  val border = "%"
  printBorder(border)
  println("Happy Birthday, Jhansi!")
  printBorder(border)
}
```

Funkcja `printBorder()`przyjmie wartość tego `border`jako dane wejściowe i wymyśli, jak wydrukować pełne obramowanie.

1. Uruchom swój kod.Twój kod nie jest wykonywany i zamiast tego widzisz ikony błędów obok kodu.



2. Spójrz na panel wyjściowy i pojawi się komunikat o błędzie.

Tak jak poprzednio, komunikat wskazuje, gdzie jest błąd i daje wskazówkę, co to może być. Ważną częścią jest: `Too many arguments for public fun printBorder()`. Wywołujesz `printBorder()`funkcję i przekazujesz obramowanie jako dane wejściowe. Jednak `printBorder()`w tej chwili definicja funkcji nie akceptuje żadnych danych wejściowych.

1. Napraw ten błąd, dodając argument dla granicy do `printBorder()`definicji funkcji. Zobacz pierwszy wiersz kodu, jak pokazano poniżej.

```
fun printBorder(border: String) {
  repeat(23) {
    print("=")
  }
  println()
}
```

- Zauważ, że nazwa argumentu to `border`.
- Po nazwie następuje dwukropek`:`
- oraz słowo `String`, które jest opisem rodzaju lub typu argumentu.

A `String`to fragment tekstu złożony ze znaków otoczonych cudzysłowami. Możesz myśleć o tym jako o koralikach ułożonych na sznurku w celu utworzenia naszyjnika, podobnie jak postacie

ułożone w kolejce, aby utworzyć słowa i tekst. Określenie, że argument musi być a, `String`pomaga systemowi wymusić, że argumentem jest tekst, a nie na przykład liczba.

1. Uruchom swój kod. Funkcja `printBorder()`przyjmuje teraz obramowanie `String`jako dane wejściowe. A kod w `main()`wywołaniach `printBorder(border)`z `border`argumentem. Twój kod powinien działać bez błędów.
2. Spójrz na wyjście swojego programu w **konsoli** i nadal pokazuje tę samą ramkę co poprzednio?

```
========================
Happy Birthday, Jhansi!
========================
```

To nie jest zamierzone zachowanie! Próbowałeś zrobić obramowanie z symbolem "%", ale program nadal drukuje obramowanie z symbolem "=". W kolejnych krokach zbadasz, dlaczego tak się dzieje.

1. W edytorze zwróć uwagę na szary wykrzyknik. Ta ikona oznacza ostrzeżenie. Ostrzeżenia dotyczą problemów z kodem, na które należy zwrócić uwagę, ale nie uniemożliwiają one uruchomienia kodu.

```
fun printBorder(border: String) {
    repeat(23) {
        print("=")
    }
    println()
}
```

2. Najedź myszą na wykrzyknik, a pojawi się komunikat. Mówi `"Parameter 'border' is never used."`To ostrzeżenie wyjaśnia problem z danymi wyjściowymi. Przekazujesz nowy ciąg dla granicy do funkcji, ale nie używasz go do drukowania.
3. Zmień `printBorder()`funkcję tak, aby używała przekazanego `border`zamiast drukowania „=”. Działa to dokładnie tak samo, jak gdyby `border`była zmienna zdefiniowana w funkcji!

```
fun printBorder(border: String) {
  repeat(23) {
    print(border)
  }
  println()
}
```

1. Uruchom kod ponownie. Dane wyjściowe powinny wyglądać tak, jak poniżej.

```
%%%%%%%%%%%%%%%%%%%%%%%
Happy Birthday, Jhansi!
%%%%%%%%%%%%%%%%%%%%%%%
```

Świetna robota, rozwiązałeś problem! Oto Twój gotowy kod.

```
fun main() {
  val border = "%"
  printBorder(border)
  println("Happy Birthday, Jhansi!")
```

```
    printBorder(border)
}

fun printBorder(border: String) {
    repeat(23) {
        print(border)
    }
    println()
}
```

Uczyniłeś `printBorder()`funkcję o wiele bardziej elastyczną, bez dodawania dużo więcej kodu. Teraz możesz wydrukować obramowanie różnych symboli z niewielką zmianą.

1. **[Opcjonalnie]** Zmieniając tylko jeden wiersz kodu w `main()`funkcji, w jaki sposób można wydrukować takie banery urodzinowe?

```
**********
Happy Birthday, Jhansi!
**********
::::::::::
Happy Birthday, Jhansi!
::::::::::
```

Zmodyfikuj funkcję tak, aby miała dwa argumenty

Co by było, gdybyś chciał użyć innego wzoru, który był dłuższy niż 1 znak, powiedzmy `"'-._.-'"`. Nie powtórzyłbyś tego wzoru 23 razy, ponieważ byłby to zbyt długi czas. Możesz to powtórzyć może 4 razy. Aby to osiągnąć, możesz zmienić liczbę powtórzeń w `repeat()`instrukcji `printBorder()`. Możesz jednak zrobić coś lepszego!

Możesz zdefiniować bardziej wyrafinowaną granicę na podstawie dwóch rzeczy:

- Wzór do powtórzenia (który już zrobiłeś)
- Ile razy chcesz powtórzyć wzór

Możesz utworzyć zmienne dla każdego, wzorca i liczby powtórzeń, a następnie przekazać obie informacje do `printBorder()`funkcji.

1. W `main()`programie zmień obramowanie na `"'-._.-'"`wzór.

```
val border = "`-._.-'"
```

1. Uruchom swój kod i zauważ, że wzorzec jest teraz zbyt długi.
2. W `main()`, poniżej definicji `border`, utwórz nową zmienną o nazwie `timesToRepeat`odpowiadającej liczbie powtórzeń. Ustaw jego wartość na 4.

```
val timesToRepeat = 4
```

1. W `main()`, podczas wywoływania `printBorder()`, dodaj liczbę powtórzeń jako drugi argument. Oddziel oba argumenty przecinkiem.

```
printBorder(border, timesToRepeat)
```

Funkcja `main()`powinna teraz wyglądać tak:

```
fun main() {
   val border = "`-._,-'"
   val timesToRepeat = 4
   printBorder(border, timesToRepeat)
   println("Happy Birthday, Jhansi!")
   printBorder(border, timesToRepeat)
}
```

Tak jak poprzednio, ten kod daje ci błąd, ponieważ masz więcej wywołań argumentów `printBorder()`niż w definicji `printBorder()`.

1. Napraw `printBorder()`, aby zaakceptować również liczbę powtórzeń jako dane wejściowe. Dodaj przecinek po argumencie, a następnie dodatkowy argument: `timesToRepeat: Int.` Pierwszy wiersz definicji funkcji wygląda teraz tak, jak pokazano poniżej.

```
fun printBorder(border: String, timesToRepeat: Int) {
```

Ogłoszenie:

- Przecinek oddziela dwa argumenty.
- `timesToRepeat`to nazwa argumentu,
- po którym następuje dwukropek : symbol,
- a type: `Int. timesToRepeat`jest liczbą, więc zamiast tworzyć type `String`, musisz utworzyć type `Int`, co jest skrótem od liczby całkowitej, liczby całkowitej.

1. Wewnątrz `printBorder()`zmień, `repeat`aby użyć `timesToRepeat`argumentu (zamiast liczby 23). Twój `printBorder()`kod powinien wyglądać tak.

```
fun printBorder(border: String, timesToRepeat: Int) {
   repeat(timesToRepeat) {
      print(border)
   }
   println()
}
```

1. Uruchom swój kod. A wynik wygląda tak, jak pokazano poniżej.

```
`-._,-'`-._,-'`-._,-'`-._,-'
Happy Birthday, Jhansi!
`-._,-'`-._,-'`-._,-'`-._,-'
```

1. Aby ten wynik był doskonały, wstaw dwie spacje na początku wiadomości z okazji urodzin. Wtedy twój wynik będzie taki, jak pokazano poniżej.

```
`-._,-'`-._,-'`-._,-'`-._,-'
 Happy Birthday, Jhansi!
`-._,-'`-._,-'`-._,-'`-._,-'
```

Oto ostateczny kod Twojego banera:

```
fun main() {
   val border = "`-._,-'"
   val timesToRepeat = 4
   printBorder(border, timesToRepeat)
   println("  Happy Birthday, Jhansi!")
   printBorder(border, timesToRepeat)
}

fun printBorder(border: String, timesToRepeat: Int) {
   repeat(timesToRepeat) {
      print(border)
   }
   println()
}
```

Gratulacje! Dzięki funkcjom, argumentom, zmiennym i pętli powtarzania nauczyłeś się podstawowych bloków konstrukcyjnych, które są używane w prawie wszystkich programach.

Zrób sobie przerwę, a następnie zajmij się kolejnym zadaniem poniżej, w którym stworzysz więcej funkcji i pętli, a zyskasz moc do zbudowania gigantycznego ciasta z odpowiednią liczbą świeczek za pomocą zaledwie kilku linijek programowania.

5. Ułóż ciasto z warstwami i świeczkami

W tym zadaniu ulepszysz kod tortu urodzinowego, aby zawsze miał odpowiedni rozmiar z odpowiednią liczbą świeczek dla każdego wieku.

- Stworzysz w sumie trzy funkcje do rysowania warstwowego tortu ze świeczkami.
- Użyjesz `repeat()`wewnątrz innego `repeat()`, tworząc coś, co nazywa się „pętlą zagnieżdżoną”.
- Sposób, w jaki zbudujesz ten kod, to sposób, w jaki możesz zbudować dowolny program, zaczynając od dużego obrazu i dodając szczegóły. Nazywa się to „rozwojem odgórnym”.
- Instrukcje nie są tak szczegółowe dla tej praktyki i możesz odwołać się do gotowego kodu, jeśli utkniesz.

Oto zdjęcie ciasta, które będziesz piec:

```
 ,,,,,,,,,,,,,,,,,,,
 |||||||||||||||||||||
==========================
@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@
```

A oto instrukcje.

Utwórz funkcję main()

1. Zresetuj swój kod w edytorze do `Hello, world!`programu.
2. Możesz usunąć argument do `main()`, ponieważ nie będziesz go używać.
3. W `main()`programie utwórz zmienną `age`i ustaw ją na 24.
4. W `main()`programie utwórz drugą zmienną `layers`i ustaw ją na 5.
5. W programie `main()`wywołaj funkcję `printCakeCandles()`i przekaż `age`. Spowoduje to błąd, ponieważ nie utworzyłeś jeszcze tej funkcji.
6. W ten sam sposób wywołaj funkcję, `printCakeTop()`a także przekaż `age`.
7. Na koniec wywołaj funkcję `printCakeBottom()`i podaj w, `age`a także liczbę `layers`.
8. Aby pozbyć się błędów, skomentuj trzy wywołania funkcji, dodając `//`na początku każdej linii, jak pokazano poniżej. Ta technika pozwala ci naszkicować kod bez wywoływania błędów.
9. Uruchom swój program i nie powinien zawierać błędów i nic nie robić.

Twoja `main()`funkcja powinna wyglądać jak poniższy kod.

```
fun main() {
  val age = 24
  val layers = 5
  // printCakeCandles(age)
  // printCakeTop(age)
  // printCakeBottom(age, layers)
}
```

Utwórz printCakeTop()

Funkcja `printCakeTop()`drukowania wierzchołka tortu, linii znaków równości, jest prawie taka sama, jak `printBorder()`funkcja, którą utworzyłeś wcześniej w tym laboratorium.

```
==========================
```

1. Poniżej `main()`funkcji dodaj pusty wiersz, a następnie utwórz funkcję, `printCakeTop()`która przyjmuje jeden argument `age`typu `Int`.
2. Wewnątrz użyj `repeat()`instrukcji, aby wydrukować jeden znak równości `age`razy plus 2. Dodatkowe dwa znaki równości mają na celu zapobieganie wypadaniu świeczek z boku tortu.
3. Na koniec, po zakończeniu `repeat()`, wydrukuj pustą linię.
4. W `main()`programie usuń dwa `//`symbole z początku wiersza kodu for `printCakeTop()`, ponieważ funkcja już istnieje.

```
printCakeTop(age)
```

Oto twoja skończona funkcja.

```
fun printCakeTop(age: Int) {
  repeat(age + 2) {
    print("=")
  }
  println()
}
```

1. Uruchom swój kod, aby zobaczyć szczyt tortu.

Utwórz printCakeCandle()

Każda świeca składa się z dwóch symboli: przecinka (,) dla płomienia i pionowej linii (|) dla korpusu świecy.

,,,,,,,,,,,,,,,,,,

||||||||||||||||||||||

Aby osiągnąć to w jednej funkcji, umieść `repeat()`w swojej funkcji dwa stwierdzenia, jedno dla płomieni, a drugie dla ciał.

1. Poniżej `main()`funkcji i `printCakeTop()`funkcji utwórz nową funkcję, `printCakeCandles()`która przyjmuje jeden argument `age`typu `Int`.
2. Wewnątrz użyj `repeat()`instrukcji, aby wydrukować jeden przecinek `,` dla płomienia.
3. Powtórz to `age`razy.
4. Na koniec wydrukuj pustą linię.
5. Dodaj oświadczenie print, aby wydrukować jedno miejsce na wstawienie świec.
6. Poniżej powtórz kroki, aby utworzyć drugą `repeat()`instrukcję, aby wydrukować korpusy świec z pionową linią `|`.
7. Na koniec wydrukuj nową linię, używając `println()`.
8. W `main()`programie usuń dwa `//`symbole z początku wiersza kodu programu `printCakeCandles()`.

```
printCakeCandles(age)
```

1. Uruchom swój kod, aby zobaczyć górę tortu i świece

Rozwiązanie:

```
fun printCakeCandles(age: Int) {
  print(" ")
  repeat(age) {
    print(",")
  }
  println() // Print an empty line

  print(" ") // Print the inset of the candles on the cake
```

```
  repeat(age) {
    print("|")
  }
  println()
}
```

Utwórz printCakeBottom()

W tej funkcji rysujesz spód ciasta o szerokości `age + 2`równej , i rysujesz je na wysokość określonej liczby warstw.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@
```

- Oznacza to, że twoja funkcja potrzebuje dwóch argumentów, jednego dla szerokości (`age`) i jednego dla wysokości (`layers`).
- Aby wydrukować spód ciasta, najpierw powtórz symbol „at” @ `age + 2`razy, aby wydrukować jedną warstwę. Następnie powtarzasz drukowanie jednej warstwy `layers`razy.

Narysuj symbol wieku +2 razy, aby utworzyć warstwę

1. Poniżej istniejących funkcji utwórz funkcję `printCakeBottom()`z dwoma argumentami `age`i `layers`, oba typu `Int`.
2. Wewnątrz funkcji użyj instrukcji, aby wydrukować jedną warstwę symboli `repeat()`„at” razy. Zakończ, drukując pustą linię, jak pokazano poniżej.`@age + 2`

```
fun printCakeBottom(age: Int, layers: Int) {
  repeat(age + 2) {
    print("@")
  }
  println()
}
```

1. Uruchom swój kod, aby sprawdzić, czy drukuje jedną linię spodu ciasta.

```
,,,,,,,,,,,,,,,,,,,,,,
||||||||||||||||||||||||
==========================
@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Zagnieżdżone instrukcje repeat()

Aby wydrukować wiele identycznych warstw spodu ciasta, możesz powiedzieć:

Dla warstwy 1 powtórz symbol 12 razy: @@@@@@@@@@@@

Dla warstwy 2 powtórz symbol 12 razy: @@@@@@@@@@@@

Dla warstwy 3 powtórz symbol 12 razy: @@@@@@@@@@@@

Możesz też powiedzieć to znacznie bardziej zwięźle, ponieważ:

Powtórz dla trzech warstw:

Repeat the symbol 12 times.

@@@@@@@@@@@@

@@@@@@@@@@@@

@@@@@@@@@@@@

To jest coś fajnego, co możesz zrobić z `repeat()`oświadczeniami. Możesz umieścić jedną `repeat()`instrukcję wewnątrz innej `repeat()`. Możesz więc utworzyć `repeat()`instrukcję w `repeat()`instrukcji, aby wydrukować symbol określoną liczbę razy dla określonej liczby warstw.

Użyj zagnieżdżonego repeat(), aby wydrukować warstwy ciasta

1. Umieść drugą `repeat()`instrukcję wokół całego kodu wewnątrz funkcji. Powtórz tę pętlę `layers`razy.
2. W `main()`programie usuń tylko te dwa `//`elementy z wiersza kodu programu `printCakeBottom()`.

```
printCakeBottom(age, layers)
```

1. Uruchom swój kod, aby zobaczyć całe ciasto.

Rozwiązanie dla `printCakeBottom()`.

```
fun printCakeBottom(age: Int, layers: Int) {
  repeat(layers) {
    repeat(age + 2) {
      print("@")
    }
    println()
  }
}
```

Gratulacje! Właśnie ukończyłeś dość złożony program z kilkoma funkcjami i zagnieżdżoną `repeat`instrukcją. A Twoje ciasto zawsze będzie miało odpowiednią ilość świeczek!

Ostateczny wynik twojego programu powinien wyglądać tak:

```
 ,,,,,,,,,,,,,,,,,,,,,,
 ||||||||||||||||||||||||
==========================
@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@
```

@@@@@@@@@@@@@@@@@@@@@@@@@@@

6. Kod rozwiązania

```
fun main() {
   val age = 24
   val layers = 5
   printCakeCandles(age)
   printCakeTop(age)
   printCakeBottom(age, layers)
}

fun printCakeCandles(age: Int) {
   print (" ")
   repeat(age) {
        print(",")
   }
   println() // Print an empty line

   print(" ") // Print the inset of the candles on the cake
   repeat(age) {
      print("|")
   }
   println()
}

fun printCakeTop(age: Int) {
   repeat(age + 2) {
      print("=")
   }
   println()
}

fun printCakeBottom(age: Int, layers: Int) {
   repeat(layers) {
      repeat(age + 2) {
         print("@")
      }
      println()
   }
}
```

7. Rozwiązywanie problemów

Jeśli oparte na przeglądarce narzędzie programistyczne Kotlin nie wykonuje Twojego kodu lub wyświetla nieoczekiwany błąd niezwiązany z Twoim kodem, możesz spróbować następujących czynności:

- Odśwież stronę za pomocą **Shift** + **Reload** .
- Poczekaj chwilę i spróbuj ponownie.

8. Podsumowanie

- Służy `${}`do otaczania zmiennych i obliczeń w tekście instrukcji drukowania. Na przykład: `${age}`gdzie `age`jest zmienną.
- Utwórz zmienną, używając `val`słowa kluczowego i nazwy. Po ustawieniu tej wartości nie można zmienić. Przypisz wartość do zmiennej za pomocą znaku równości. Przykładami wartości są tekst i liczby.
- A `String`to tekst otoczony cudzysłowami, np `"Hello"`. .
- An `Int`to całkowita liczba dodatnia lub ujemna, na przykład 0, 23 lub -1024.
- Do funkcji można przekazać jeden lub więcej argumentów, których funkcja ma używać, na przykład:`fun printCakeBottom(age:Int, layers:Int) {}`
- Użyj `repeat() {}`instrukcji, aby kilkakrotnie powtórzyć zestaw instrukcji. Na przykład: `repeat (23) { print("%") }`lub`repeat (layers) { print("@@@@@@@@@@") }`
- Pętla to instrukcja wielokrotnego powtarzania instrukcji . Instrukcja jest `repeat()`przykładem pętli.
- Możesz zagnieżdżać pętle, to znaczy umieszczać pętle w pętlach. Na przykład, możesz utworzyć `repeat()`instrukcję w `repeat()`instrukcji, aby wydrukować symbol wiele razy dla wielu wierszy, tak jak zrobiłeś to dla warstw ciasta.

Podsumowanie używania argumentów funkcji: Aby użyć argumentów z funkcją, musisz zrobić trzy rzeczy:

- Dodaj argument i typ do definicji funkcji:`printBorder(border: String)`
- Użyj argumentu wewnątrz funkcji:`println(border)`
- Podaj argument, gdy wywołujesz funkcję:`printBorder(border)`

9. Dowiedz się więcej

- https://developer.android.com/training/kotlinplayground
- Słownictwo dla Androida Podstawy w Kotlin

Oto oficjalna dokumentacja koncepcji Kotlin, których nauczyłeś się w tym laboratorium.

- Definiowanie zmiennych
- Uwagi
- Definiowanie funkcji
- `repeat`oświadczenie

Ścieżka 2. Stwórz swoją pierwszą aplikację na Androida

Pobierz i zainstaluj Android Studio

1. Zanim zaczniesz

Podczas tego ćwiczenia z kodowania dowiesz się, jak pobrać i zainstalować Android Studio, środowisko programistyczne Google Android.

Warunki wstępne

- Średnie umiejętności obsługi komputera, w tym rozumienie struktury plików i używanie umiarkowanie złożonych aplikacji, takich jak arkusz kalkulacyjny, edytor tekstu i edytor zdjęć.
- Umiejętności nawigacji komputerowej, aby można było otwierać i dostosowywać ustawienia oraz identyfikować wersje przeglądarki i systemu operacyjnego (OS).
- Możliwość weryfikacji wymagań systemowych (wymagania dotyczące miejsca na dysku i pamięci oraz rozdzielczości ekranu) oraz pobierania, instalowania i aktualizowania oprogramowania.

Czego się nauczysz

- Jak sprawdzić, czy konfiguracja komputera spełnia minimalne wymagania do uruchomienia narzędzia Android Studio.
- Jak pobrać i zainstalować narzędzie Android Studio.

Czego potrzebujesz

- Komputer z najnowszą wersją systemu Windows, Linux lub macOS.
- Dostęp do Internetu dla Twojego komputera.

2. Poznaj Android Studio

Android Studio jest oficjalnym zintegrowanym środowiskiem programistycznym (IDE) do tworzenia aplikacji na Androida zbudowanym i dystrybuowanym przez Google. To specjalistyczne warsztaty z narzędziami, które ułatwiają programistom projektowanie, tworzenie, uruchamianie i testowanie aplikacji na platformę Android. Android Studio wykorzystuje IntelliJ IDEA jako podstawę i zawiera preinstalowaną wtyczkę Androida wraz z kilkoma poprawkami specjalnie dla platformy Android, więc będzie Ci się wydawać bardzo znajomy.

W tym ćwiczeniu z kodowania zainstalujesz Android Studio.

Android

Android to system operacyjny (taki jak Windows, Linux lub macOS) dla smartfonów i innych urządzeń, takich jak tablety, urządzenia do noszenia, telewizory i samochody. Możesz uruchamiać aplikacje na Androida, takie jak Telefon, Wiadomości, Gmail, Zdjęcia i wszystkie swoje gry.

Wskazówka: Android to jeden z najpopularniejszych mobilnych systemów operacyjnych. Na całym świecie jest ponad 2,5 miliarda urządzeń, w tym zegarków, telewizorów i samochodów, z systemem Android.

3. Sprawdź wymagania systemowe

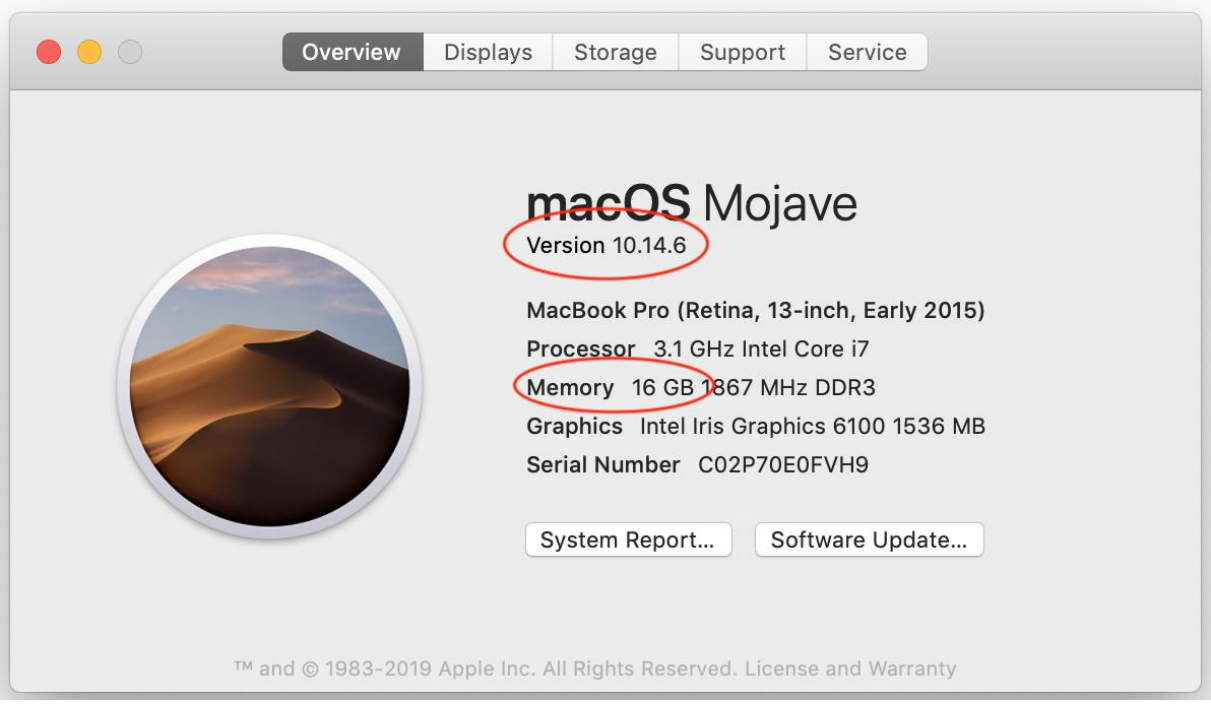
Aby uruchomić Android Studio, Twój komputer i system operacyjny muszą spełniać pewne wymagania.

1. Upewnij się, że masz dobre i stabilne połączenie internetowe. Czas instalacji może się różnić w zależności od szybkości i niezawodności połączenia internetowego.
2. Otwórz dowolną przeglądarkę internetową, taką jak Chrome , i przejdź do https://developer.android.com/studio#Requirements . Ta strona zawiera szczegółowe wymagania sprzętowe, które są potrzebne do zainstalowania i uruchomienia Android Studio na Twoim komputerze.
3. Wykonaj poniższe czynności, aby sprawdzić wymagania systemowe dla systemu Windows lub macOS.

System operacyjny Mac

1. Wybierz **menu Apple**  -> **O tym Macu** .
2. W wyskakującym okienku, na karcie **Przegląd** , poszukaj numeru wersji systemu operacyjnego i upewnij się, że mieści się w wymaganym zakresie.
3. Sprawdź obok opcji **Pamięć** , czy podana całkowita pamięć spełnia lub przekracza wymagane minimum.

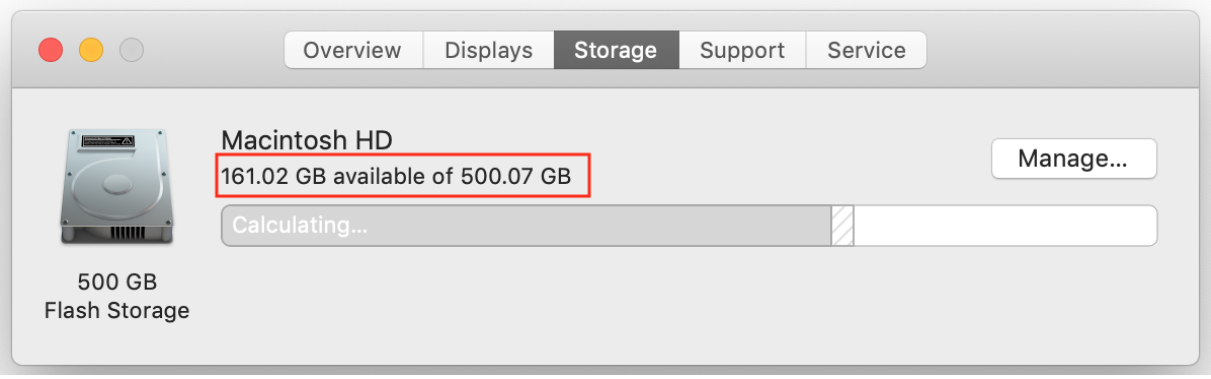
Na przykład na poniższym zrzucie ekranu wersja systemu operacyjnego to **10.14.6** , a pamięć to 16 GB.



1. W tym samym wyskakującym okienku przejdź do karty **Wyświetlacze** .
2. W opisie wyświetlacza upewnij się, że rozdzielczość ekranu komputera odpowiada lub przekracza zalecaną rozdzielczość. Na poniższym zrzucie ekranu rozdzielczość wynosi **2560 X 1600** .

1. Przejdź do zakładki **Pamięć .**

2. Sprawdź dostępne miejsce na dysku i upewnij się, że spełnia lub przekracza zalecane miejsce na dysku do uruchomienia Android Studio.



1. Jeśli Twój komputer spełnia wszystkie wymagania, przejdź do Pobierz Android Studio .

Windows 10

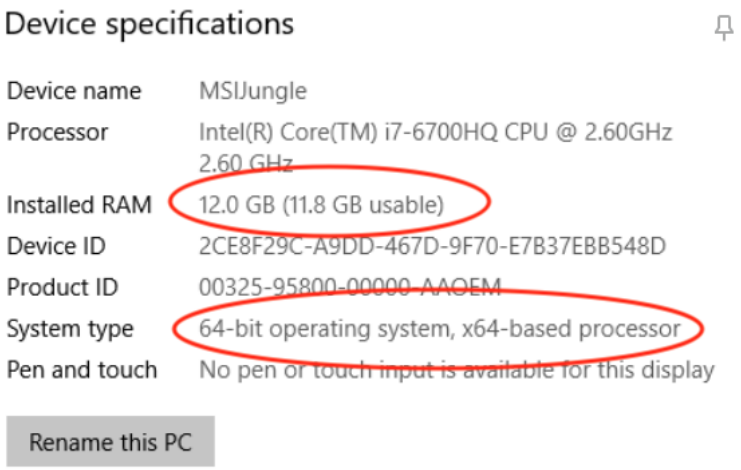
Na komputerze z systemem Windows wszystkie informacje potrzebne do zweryfikowania wymagań systemowych można znaleźć w **Ustawieniach** .

1. Otwórz **Ustawienia** .

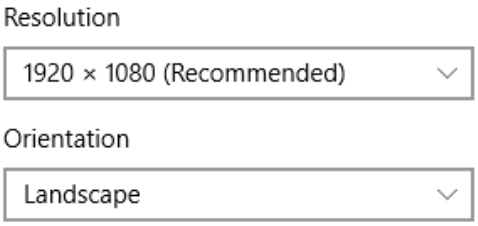
Wskazówka: możesz użyć narzędzia **wyszukiwania** obok przycisku **Start** u dołu ekranu, aby go zlokalizować.

1. Kliknij **System** .
2. U dołu lewego okienka nawigacyjnego kliknij **Informacje** .
3. Upewnij się, że **specyfikacje systemu Windows** spełniają lub przekraczają wymagania.

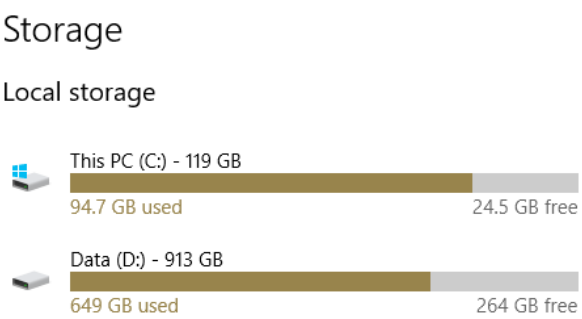
1. Wybierz **Specyfikacje urządzenia.**



1. Upewnij się, że **zainstalowanej pamięci RAM** jest co najmniej tyle, ile jest wymagane, a **typ systemu** to 64-bitowa wersja systemu operacyjnego.
2. W nawigacji po lewej stronie kliknij **Wyświetl** .
3. Upewnij się, że **rozdzielczość** jest taka sama lub lepsza niż wymagana.



1. W okienku nawigacji po lewej stronie kliknij **Pamięć** .
2. Upewnij się, że **lokalna pamięć** ma wystarczająco dużo wolnego miejsca, aby zainstalować Android Studio.



4. Pobierz Studio Android

Uwaga : chociaż wygląd okien dialogowych instalacji i ikon może się zmieniać w różnych wersjach Android Studio, kroki instalacji powinny zasadniczo pozostać takie same.

Pobierz plik instalacyjny Android Studio

1. Otwórz dowolną przeglądarkę internetową i przejdź do https://developer.android.com/studio .

To jest witryna Android Developers, z której możesz pobrać Android Studio. Ta strona automatycznie wykrywa Twój system operacyjny.

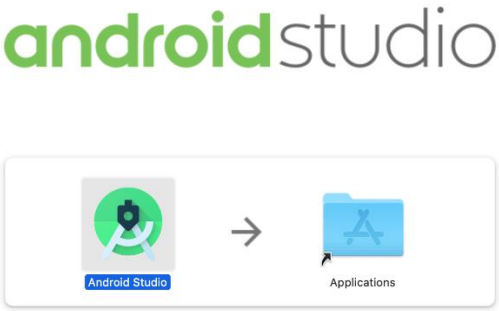
1. Kliknij **Pobierz Android Studio** . Otworzy się strona **Regulamin** z umową licencyjną Android Studio.
2. Przeczytaj umowę licencyjną.
3. Na dole strony, jeśli zgadzasz się z warunkami, zaznacz „ **Przeczytałem i zgadzam się z powyższymi warunkami** ”.
4. Kliknij **Pobierz Android Studio dla...** , aby rozpocząć pobieranie.
5. Po wyświetleniu monitu zapisz plik w lokalizacji, w której możesz go łatwo zlokalizować (na przykład w folderze **Pulpit** lub **Pobrane).**
6. Poczekaj na zakończenie pobierania. Może to chwilę potrwać i może być dobrym momentem na napicie się herbaty!

5. Zainstaluj Android Studio

Uwaga : jeśli potrzebujesz dodatkowej pomocy lub chcesz dostosować instalację, zapoznaj się z instrukcjami instalacji Android Studio, która zawiera screencasty.

Zainstaluj Android Studio na macOS

1. Otwórz folder, do którego pobrałeś i zapisałeś plik instalacyjny Android Studio.
2. Kliknij dwukrotnie pobrany plik. Wyświetlane są następujące wyskakujące okienka:



1. Przeciągnij i upuść ikonę **Android Studio** do folderu **Aplikacje .**
2. W folderze **Aplikacje** kliknij dwukrotnie ikonę **Android Studio** , aby uruchomić **Kreatora instalacji Android Studio.**

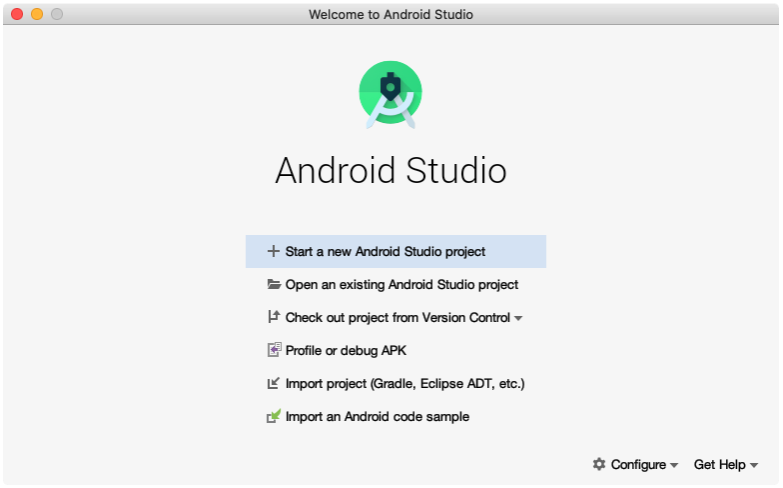
Jeśli zobaczysz ostrzeżenie o instalacji lub uruchomieniu pliku pobranego z Internetu, zaakceptuj instalację.

Postępuj zgodnie z **kreatorem konfiguracji Android Studio** i zaakceptuj ustawienia domyślne dla wszystkich kroków.

Podczas instalacji kreator instalacji pobiera i instaluje dodatkowe komponenty i narzędzia potrzebne do tworzenia aplikacji na Androida. Może to zająć trochę czasu, w zależności od szybkości połączenia internetowego. Więc możesz uzupełnić swoją filiżankę herbaty!

1. Po zakończeniu instalacji Android Studio uruchomi się automatycznie.

Otworzy się okno dialogowe **Witamy w Android Studio** i możesz zacząć tworzyć aplikacje!



Zainstaluj Android Studio w systemie Windows

1. Otwórz folder, do którego pobrałeś i zapisałeś plik instalacyjny Android Studio.
2. Kliknij dwukrotnie pobrany plik.

Jeśli zobaczysz ostrzeżenie o zezwoleniu instalacji na wprowadzenie zmian na komputerze, potwierdź instalację.

Zostanie wyświetlone okno dialogowe **Witamy w konfiguracji Android Studio** .

1. Kliknij **Dalej** , aby rozpocząć instalację.
2. Zaakceptuj domyślne ustawienia instalacji dla wszystkich kroków.
3. Po zakończeniu instalacji kliknij przycisk **Zakończ .**

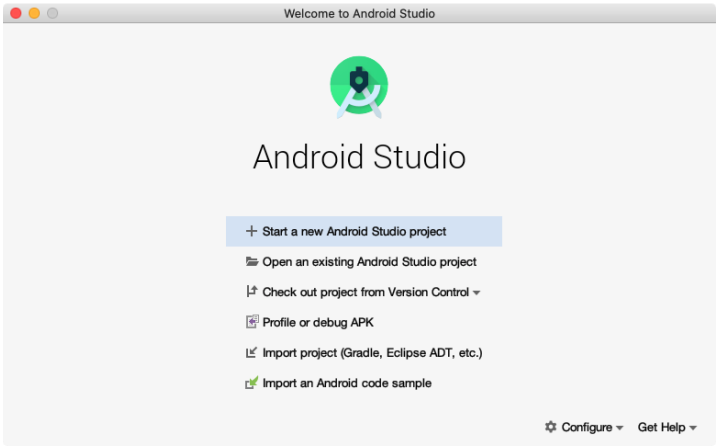
Wyświetli się **kreator instalacji Android Studio.**

1. Postępuj zgodnie z **kreatorem konfiguracji Android Studio** i zaakceptuj ustawienia domyślne dla wszystkich kroków.

Podczas instalacji kreator instalacji pobiera i instaluje dodatkowe komponenty i narzędzia potrzebne do tworzenia aplikacji na Androida. Może to zająć trochę czasu, w zależności od szybkości połączenia internetowego. Więc możesz uzupełnić swoją filiżankę herbaty!

1. Po zakończeniu pobierania i instalacji kliknij przycisk **Zakończ** .

Zostanie wyświetlone okno dialogowe **Witamy w Android Studio** i możesz zacząć tworzyć aplikacje!



Gratulacje! Pomyślnie zainstalowałeś Android Studio. Teraz jesteś gotowy na kolejny krok!

6. Dowiedz się więcej

- Słownictwo dla Androida Podstawy w Kotlin

Dokumentacja dla programistów Androida:

- Pobierz Studio Android
- Zainstaluj Android Studio
- Poznaj Android Studio

Utwórz i uruchom swoją pierwszą aplikację na Androida

1. Wstęp

W tym ćwiczeniu z programowania utworzysz swoją pierwszą aplikację na Androida (Wszystkiego najlepszego), zaczynając od szablonu podstawowej aplikacji dostarczonej przez Android Studio. Dowiesz się również, jak wygląda projekt na Androida i dowiesz się, jak korzystać z różnych okien w Android Studio.

Warunki wstępne

- Znajomość konfiguracji, konfiguracji i korzystania z aplikacji, takich jak edytor tekstu lub arkusz kalkulacyjny

Czego się nauczysz

- Jak utworzyć projekt Android Studio dla aplikacji na Androida za pomocą szablonu?

Co zbudujesz

- Podstawowa aplikacja na Androida z szablonu

Czego potrzebujesz

- Komputer z zainstalowanym Android Studio

2. Stwórz swoją pierwszą aplikację

W tym zadaniu utworzysz aplikację na Androida, korzystając z szablonu projektu dostarczonego przez Android Studio.

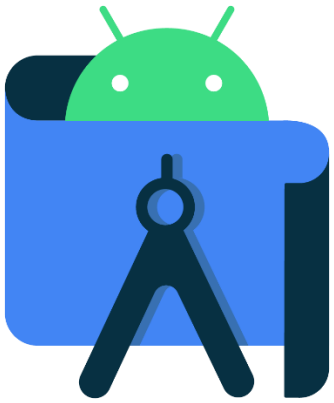
Szablony projektów

W Android Studio szablon projektu to aplikacja dla systemu Android, która zawiera wszystkie niezbędne części, ale niewiele robi. Ma to na celu pomóc Ci szybciej rozpocząć pracę i zaoszczędzić trochę pracy. Niektóre przykłady szablonów w Android Studio to aplikacja z mapą i aplikacja z wieloma ekranami.

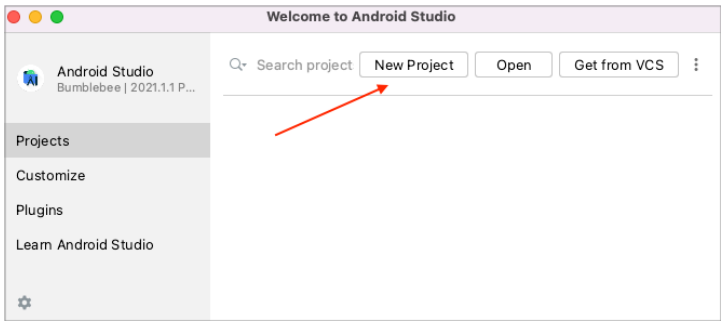
Utwórz projekt pustej aktywności

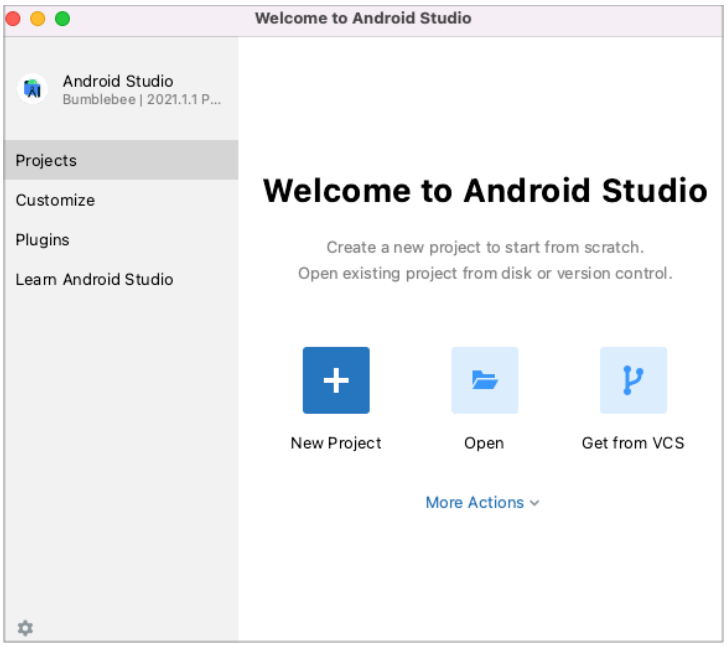
W tych krokach utworzysz nowy projekt Android Studio, korzystając z szablonu projektu **Empty Activity** dla nowej aplikacji.

1. Uruchom Android Studio (jeśli nie jest jeszcze otwarte), klikając ikonę Android Studio:

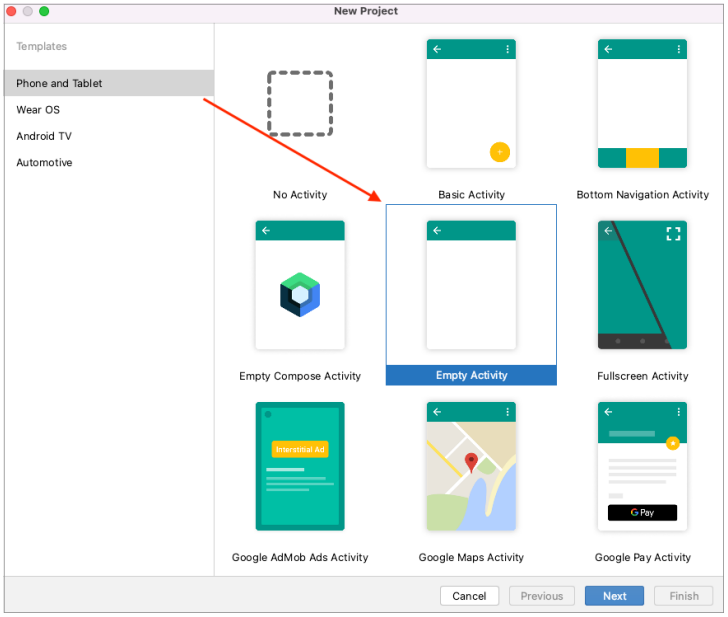


1. Otworzy się okno Witamy **w Android Studio .** Kliknij **Nowy projekt** .





1. Otworzy się okno **Nowy projekt** z listą szablonów dostarczonych przez Android Studio.

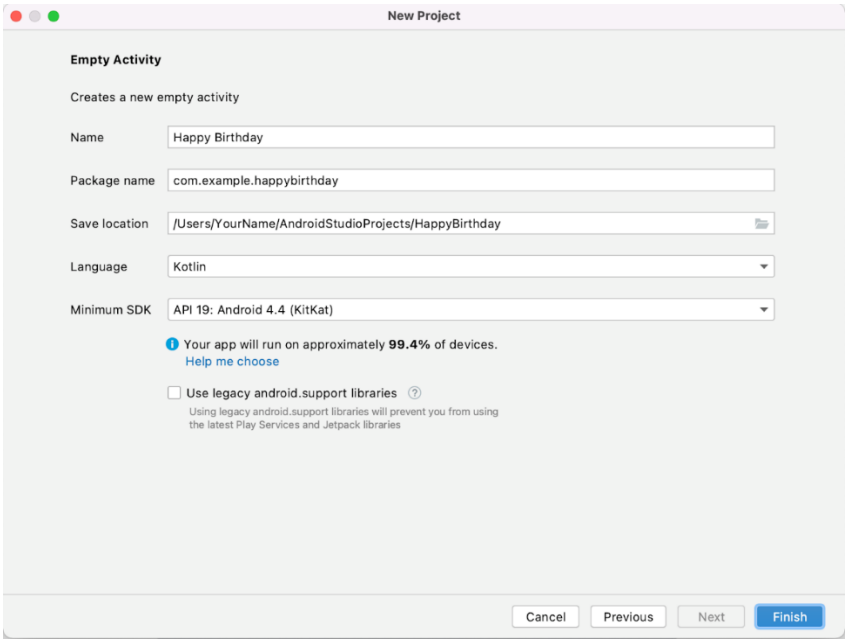


1. Kliknij zakładki u góry i przewiń szablony, aby dowiedzieć się, co możesz zrobić! Dostępne są szablony dla wielu różnych typów urządzeń (takich jak telefony, tablety i zegarki) oraz różnych typów aplikacji (aplikacje z przewijanymi ekranami, aplikacje z mapami i aplikacje z wymyślną nawigacją).

2. W lewym górnym rogu okna kliknij kartę **Telefon i tablet** .
3. W górnym wierszu kliknij szablon **Puste działanie** , aby wybrać go jako szablon dla swojego projektu.

Szablon **pustej aktywności** to najprostszy szablon, którego można użyć do stworzenia aplikacji. Ma pojedynczy ekran i wyświetla proste „Hello World!” wiadomość.

1. U dołu okna kliknij **Dalej** . Otworzy się okno dialogowe **Nowy projekt .**

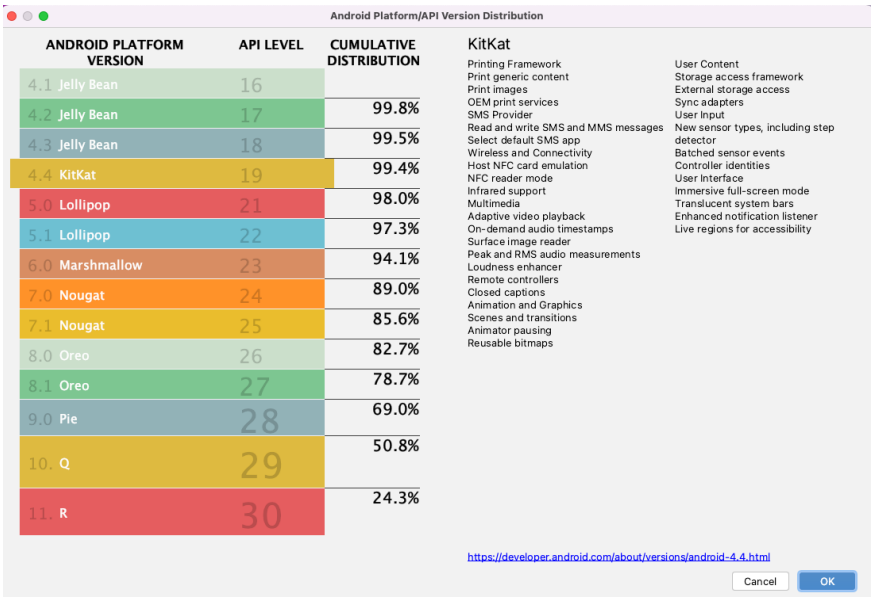


Skonfiguruj swój projekt w następujący sposób:

1. **Nazwa** to nazwa Twojej aplikacji. W polu poniżej **Nazwa** wprowadź `Happy Birthday`nazwę swojego projektu.
2. **Nazwa pakietu** to nazwa używana przez system Android do jednoznacznej identyfikacji Twojej aplikacji. Zwykle domyślnie jest to nazwa Twojej organizacji, po której następuje nazwa aplikacji, wszystkie pisane małymi literami (w tym przypadku „ `com.example.happybirthday`”).
3. **Lokalizacja zapisu** to lokalizacja, w której zapisywane są wszystkie pliki związane z Twoim projektem. Zanotuj, gdzie to jest na Twoim komputerze, aby móc znaleźć swoje pliki. Możesz opuścić lokalizację zapisu również tak, jak na razie.
4. **Język** definiuje język programowania, którego chcesz użyć w swoim projekcie. Upewnij się, że **język** to `Kotlin`.
5. **Minimalny pakiet SDK** wskazuje minimalną wersję Androida, na której może działać Twoja aplikacja. Wybierz `API 19: Android 4.4 (KitKat)`z listy rozwijanej.

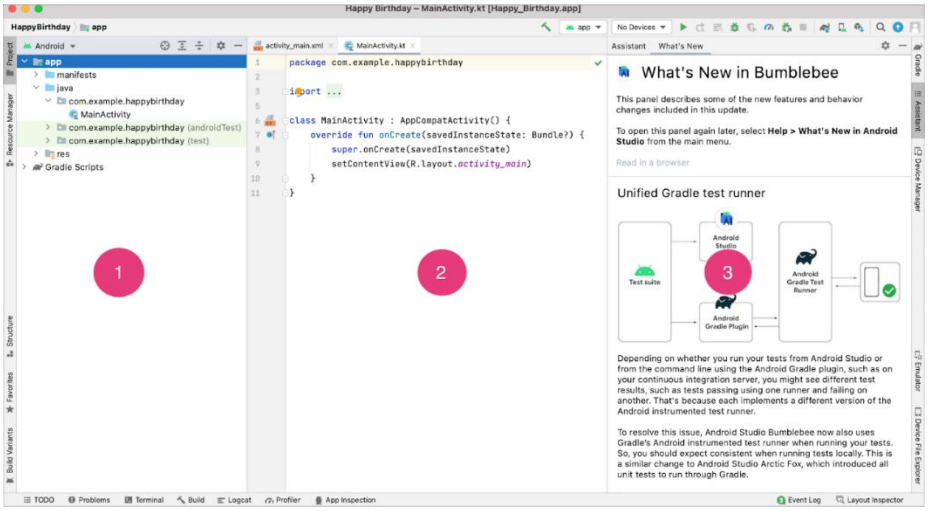
Uwaga: istnieje wiele różnych wersji systemu operacyjnego Android, z których każda ma nazwę w kolejności alfabetycznej, gdy jest publikowana.

1. Poniżej **Minimalny pakiet SDK** , zwróć uwagę na informację o tym, na ilu urządzeniach Twoja aplikacja może działać na wybranym poziomie interfejsu API. Jeśli jesteś ciekawy, kliknij link **Pomóż mi wybrać** , aby wyświetlić listę różnych wersji Androida, jak pokazano poniżej. Następnie wróć do okna **Nowy projekt** .



1. W oknie **Nowy projekt** upewnij się, że pole **Użyj starszych bibliotek android.support** nie jest zaznaczone. Kliknij znak zapytania, jeśli chcesz dowiedzieć się więcej na ten temat.
2. Kliknij **Zakończ** .

Android Studio otwiera projekt i wszystkie jego pliki.



Gdy po raz pierwszy otworzysz Android Studio, zobaczysz trzy okna:

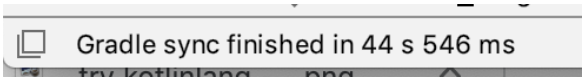
(1) Okno **Projekt** pokazuje pliki i foldery Twojego projektu.

(2) Okno **edycji** służy do edycji kodu.

(3) W oknie **Co nowego** wyświetlane są wiadomości i przydatne wskazówki.

W prawym dolnym rogu Android Studio pasek postępu lub komunikat wskazuje, czy Android Studio nadal pracuje nad konfiguracją twojego projektu. Na przykład:

1. Poczekaj, aż Android Studio zakończy konfigurowanie projektu. Wiadomość w lewym dolnym rogu, taka jak pokazana poniżej, poinformuje Cię o zakończeniu projektu.



3. Uruchom aplikację na urządzeniu wirtualnym (emulatorze)

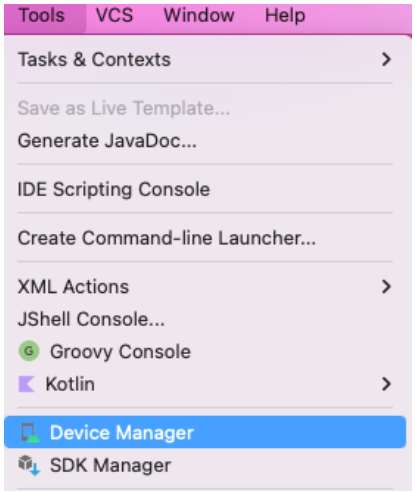
W tym zadaniu użyjesz Menedżera urządzeń, aby utworzyć wersję oprogramowania (emulator) urządzenia mobilnego i uruchomić go na swoim komputerze. Urządzenie wirtualne lub emulator symuluje konfigurację określonego typu urządzenia z systemem Android, takiego jak telefon. Może to być dowolny telefon lub tablet z wybraną przez Ciebie wersją systemu Android. Następnie użyjesz urządzenia wirtualnego, aby uruchomić aplikację utworzoną za pomocą szablonu **Puste działanie .**

Uwaga: Emulator Androida jest niezależną aplikacją służącą do konfigurowania urządzenia wirtualnego i ma własne wymagania systemowe. Urządzenia wirtualne mogą zajmować dużo miejsca na dysku. Jeśli napotkasz jakiekolwiek problemy, zobacz Uruchamianie aplikacji w emulatorze systemu Android .

Utwórz urządzenie wirtualne z systemem Android (AVD)

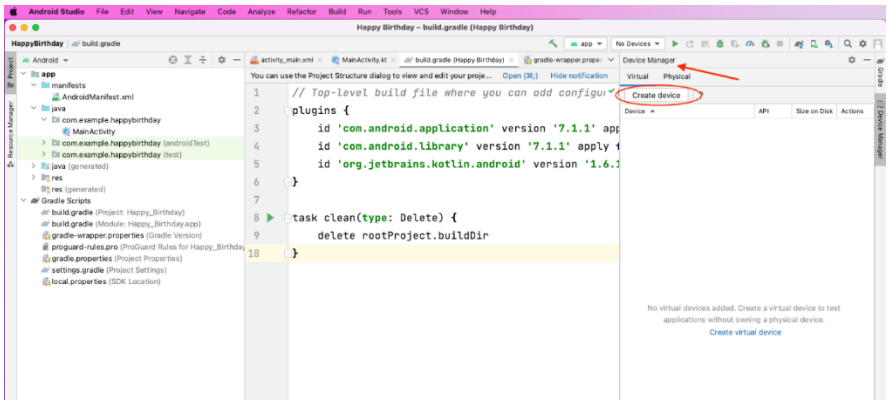
Pierwszym krokiem do uruchomienia emulatora na komputerze jest utworzenie konfiguracji dla urządzenia wirtualnego.

1. Z paska menu Android Studio wybierz **Narzędzia > Menedżer urządzeń** .



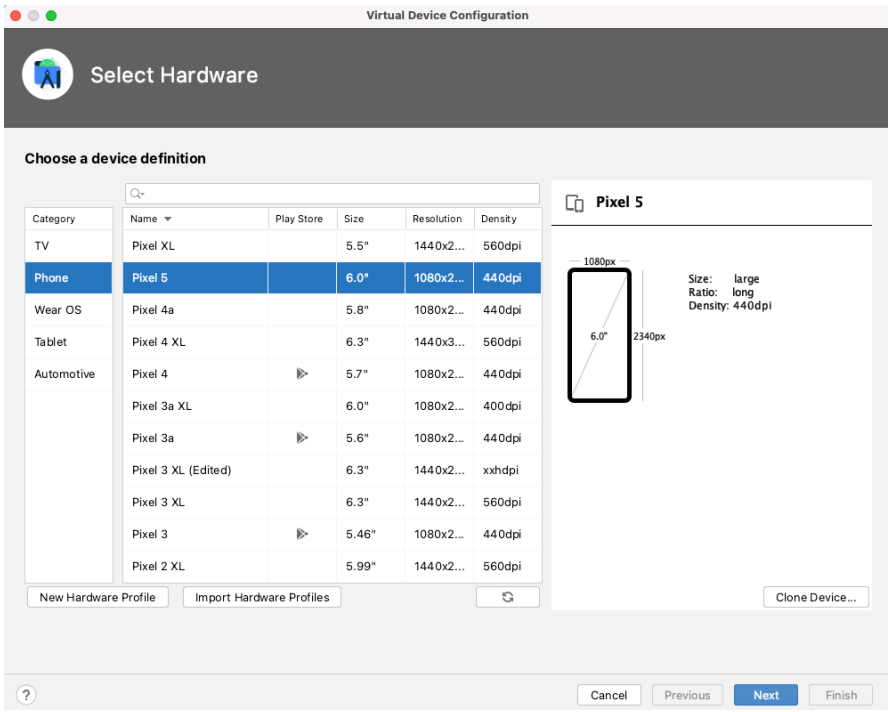
Wskazówka: Możesz także otworzyć Menedżera urządzeń, klikając jego ikonę na pasku narzędzi.

Menedżer **urządzeń** wyświetla się, jak pokazano poniżej. (Jeśli wcześniej utworzyłeś urządzenie, będzie ono tutaj wymienione).



1. Kliknij **Utwórz urządzenie** .

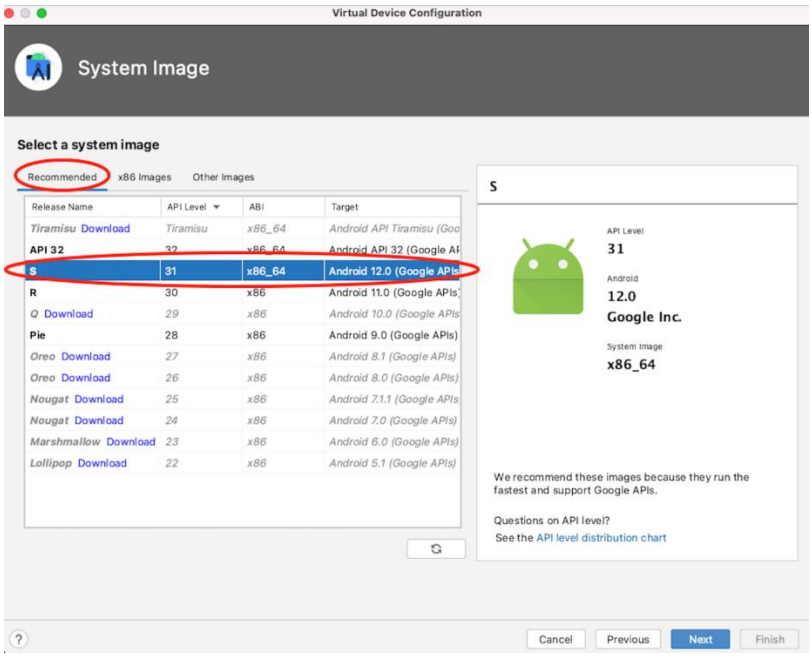
Pojawi się okno **Wybierz sprzęt** .



W oknie **Wybierz sprzęt** wyświetlana jest lista wstępnie skonfigurowanych urządzeń uporządkowanych według **kategorii** , z których można wybierać. Dla każdego urządzenia tabela zawiera kolumnę określającą jego rozmiar wyświetlacza (**Rozmiar**), rozdzielczość ekranu w pikselach (**Rozdzielczość**) i gęstość pikseli (**Gęstość**).

1. Wybierz **Telefon** jako kategorię.
2. Wybierz telefon (na przykład a Pixel 5), a następnie kliknij **Dalej** . Możesz wybrać dowolny telefon, ale do tego ćwiczenia z programowania wybierz nowsze urządzenie.

Pojawi się okno **Obraz systemu .** W tym miejscu wybierasz wersję systemu Android do uruchomienia na urządzeniu wirtualnym. Dzięki temu możesz przetestować swoją aplikację w różnych wersjach systemu Android.



1. Na karcie **Zalecane** wybierz wersję systemu Android do uruchomienia na urządzeniu wirtualnym. W chwili pisania tego tekstu była to najnowsza wersja Androida, ale możesz wybrać dowolną późniejszą stabilną wersję. Sprawdź tutaj listę stabilnych wersji.

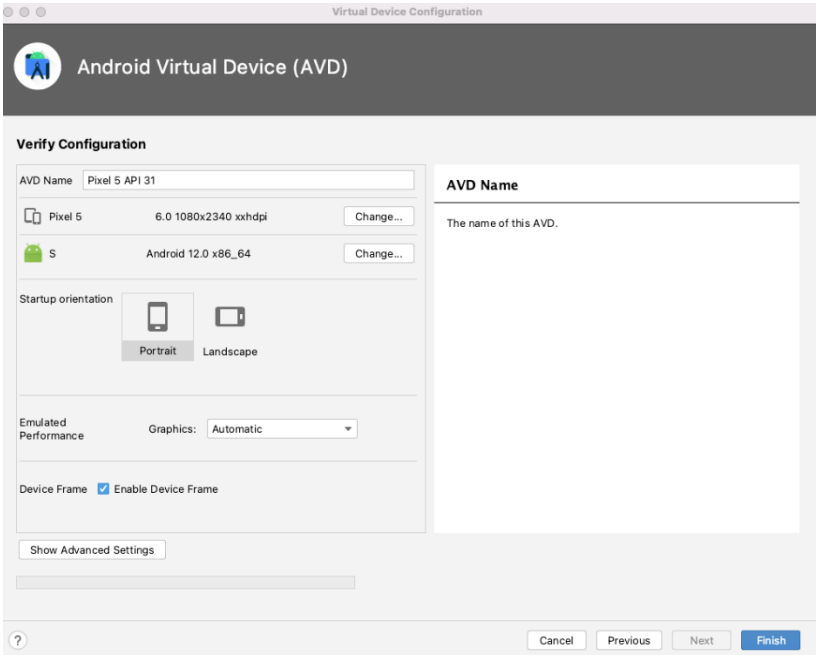
Uwaga: jeśli łącze **pobierania** jest widoczne obok obrazu systemu, którego chcesz użyć, oznacza to, że ten obraz nie jest zainstalowany na komputerze. Musisz zainstalować obraz przed skonfigurowaniem urządzenia wirtualnego.

Aby zainstalować obraz systemu, kliknij łącze **Pobierz** . Pamiętaj, że pobieranie może zająć dużo czasu, w zależności od połączenia internetowego. Po zakończeniu pobierania kliknij **Zakończ** .

Ważne: te obrazy systemu Android zajmują dużo miejsca na dysku, więc tylko kilka z nich jest częścią oryginalnej instalacji. Dostępnych jest znacznie więcej wersji systemu Android niż pokazano w zakładce **Rekomendowane .** Aby je zobaczyć, spójrz na karty **Obrazy x86** i **Inne obrazy .**

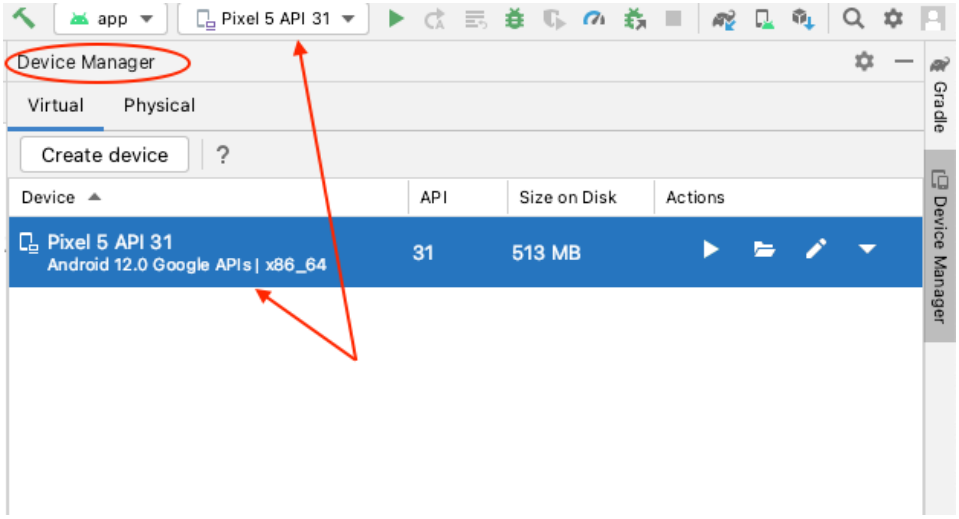
1. Kliknij **Dalej** .

Zostanie wyświetlone okno **Android Virtual Device (AVD)** , w którym można wybrać dodatkowe szczegóły konfiguracji urządzenia.

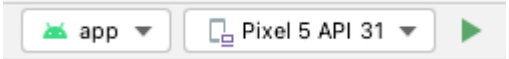


1. W polu **Nazwa AVD** wprowadź nazwę urządzenia wirtualnego z systemem Android. Resztę pozostaw bez zmian.
2. Kliknij **Zakończ** .

Twoje nowe urządzenie wirtualne zostanie wyświetlone w oknie **Menedżera urządzeń** i będzie gotowe do użycia.



Uruchom swoją aplikację na urządzeniu wirtualnym

1. Jeśli jeszcze nie jest, przenieś Android Studio na pierwszy plan.
2. W Android Studio na pasku narzędzi znajdź rozwijane menu urządzenia wirtualnego (będzie ono wyglądać podobnie do poniższego przykładu), a następnie wybierz utworzone urządzenie wirtualne z listy rozwijanej. 

(Lub kliknij **Uruchom > Wybierz urządzenie...** , a następnie wybierz urządzenie wirtualne z dostępnych urządzeń w wyskakującym okienku.)

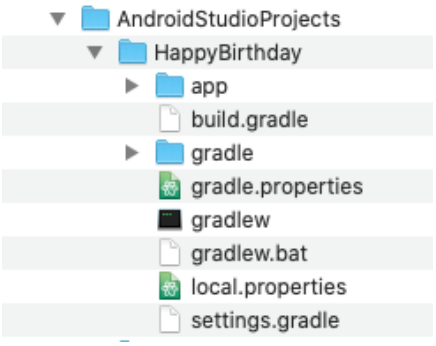
3. W Android Studio wybierz **Uruchom > Uruchom aplikację** lub kliknij ikonę **Uruchom** na pasku narzędzi. Urządzenie wirtualne uruchamia się i uruchamia tak samo jak urządzenie fizyczne. W zależności od szybkości komputera może to zająć trochę czasu.

Gdy Twoja aplikacja jest gotowa, otwiera się na urządzeniu wirtualnym, jak pokazano poniżej.

Dobra robota! Twoje urządzenie wirtualne jest teraz uruchomione. Zwróć uwagę, że tytuł to teraz „ Happy Birthday”, a Hello World!na ekranie wyświetla się „ ”.

4. Znajdź swoje pliki projektu

Po skonfigurowaniu projektu Android Studio utworzyło na komputerze folder dla wszystkich projektów Androida o nazwie **AndroidStudioProjects** . Wewnątrz folderu **AndroidStudioProjects** Android Studio tworzy również folder o tej samej nazwie, co Twoja aplikacja (w tym przypadku **HappyBirthday).**



Folder HappyBirthday to folder Twojego projektu. Android Studio zapisuje zarówno pliki, które tworzysz, jak i pliki tworzone przez Android Studio w folderze projektu.

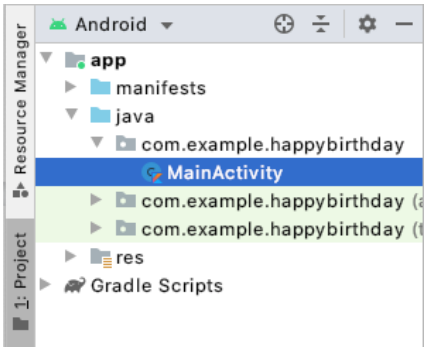
1. W Android Studio spójrz na okno **projektu** po lewej stronie. Okno **projektu** pokazuje pliki i foldery twojego projektu.

Pliki w oknie **projektu** są zorganizowane w celu ułatwienia nawigacji między plikami projektu podczas pisania kodu. Jeśli jednak spojrzysz na pliki w przeglądarce plików, takiej jak Finder lub Eksplorator Windows, hierarchia plików jest zorganizowana bardzo inaczej.

W tym zadaniu poznasz te dwa różne widoki hierarchii folderów projektu.

1. W Android Studio, w oknie **Projekt** , wybierz **Android** z rozwijanego menu w lewym górnym rogu.

Powinieneś zobaczyć listę plików podobną do następującej:

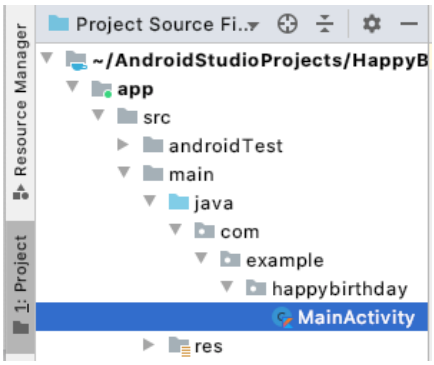


Ten widok i organizacja plików jest przydatna podczas pracy nad pisaniem kodu do projektu.

Możesz także przeglądać pliki tak, jak wyglądałyby w przeglądarce plików, takiej jak Finder (w systemie macOS) lub Eksplorator (w systemie Windows).

1. W oknie **Projekt** wybierz z menu rozwijanego **Pliki źródłowe projektu .**

Zauważ, że tytuł zmienia się na folder, w którym przechowywane są pliki projektu.



Możesz teraz przeglądać pliki w taki sam sposób, jak w dowolnym eksploratorze plików.

1. Aby wrócić do poprzedniego widoku, w oknie **Projekt** ponownie wybierz **Android** .

Świetny! Teraz możesz tworzyć i uruchamiać aplikację z szablonu, a także znajdować pliki projektu.

5. Podsumowanie

- Aby utworzyć nowy projekt, uruchom Android Studio, kliknij **+ Rozpocznij nowy projekt Android Studio** , nazwij swój projekt, wybierz szablon i wypełnij szczegóły.

- Aby utworzyć urządzenie wirtualne z systemem Android (emulator) do uruchamiania aplikacji, wybierz **Narzędzia > Menedżer urządzeń,** a następnie użyj Menedżera urządzeń, aby wybrać urządzenie sprzętowe i obraz systemu.
- Aby uruchomić aplikację na urządzeniu wirtualnym, upewnij się, że utworzono urządzenie, wybierz urządzenie z menu rozwijanego paska narzędzi, a następnie uruchom aplikację, klikając ikonę **Uruchom**  na pasku narzędzi.
- Aby znaleźć pliki projektu, w oknie **Projekt** wybierz z listy rozwijanej **Pliki źródłowe projektu .**

6. Dowiedz się więcej

- Słownictwo dla Androida Podstawy w Kotlin
- Poznaj Android Studio
- Przegląd projektów
- Utwórz projekt
- Dodaj kod z szablonu
- Zbuduj i uruchom swoją aplikację
- Uruchamiaj aplikacje na emulatorze Androida

Opcjonalny: Uruchom aplikację na urządzeniu mobilnym

1. Zanim zaczniesz

Warunki wstępne

- Podstawowe zrozumienie korzystania z Android Studio.
- Możliwość otwierania i dostosowywania ustawień na urządzeniu z Androidem.

Czego się nauczysz

- Jak umożliwić urządzeniu z Androidem uruchamianie aplikacji z Android Studio.
- Jak podłączyć i uruchomić aplikację w Android Studio na fizycznym urządzeniu z Androidem.

Czego potrzebujesz

- Android Studio został pobrany i zainstalowany na Twoim komputerze.
- Projekt aplikacji skonfigurowany w Android Studio.
- Urządzenie z Androidem, takie jak telefon lub tablet.
- Kabel USB do podłączenia urządzenia z systemem Android do komputera za pośrednictwem ich portów USB.

Uwaga : Sprawdź ten artykuł , jeśli potrzebujesz pomocy w określeniu rodzaju portów USB komputera i urządzenia z systemem Android oraz odpowiedniego kabla.

2. Włącz debugowanie USB

Aby umożliwić Android Studio komunikację z urządzeniem z systemem Android, musisz włączyć debugowanie USB w ustawieniach **opcji programisty** urządzenia.

Aby wyświetlić opcje programisty i włączyć debugowanie USB:

1. Na urządzeniu z Androidem otwórz **Ustawienia** i wyszukaj **Informacje o telefonie** .
2. Stuknij opcję **Informacje o telefonie** , a następnie siedem razy stuknij **Numer kompilacji** . Wprowadź hasło lub kod PIN urządzenia, jeśli zostaniesz o to poproszony.
3. Wróć do **Ustawień** i dotknij **System** . **Opcje programisty** powinny teraz pojawić się na liście. Być może trzeba będzie otworzyć Opcje **zaawansowane** , aby go znaleźć.
4. Stuknij **Opcje programisty** , a następnie włącz **debugowanie USB** .

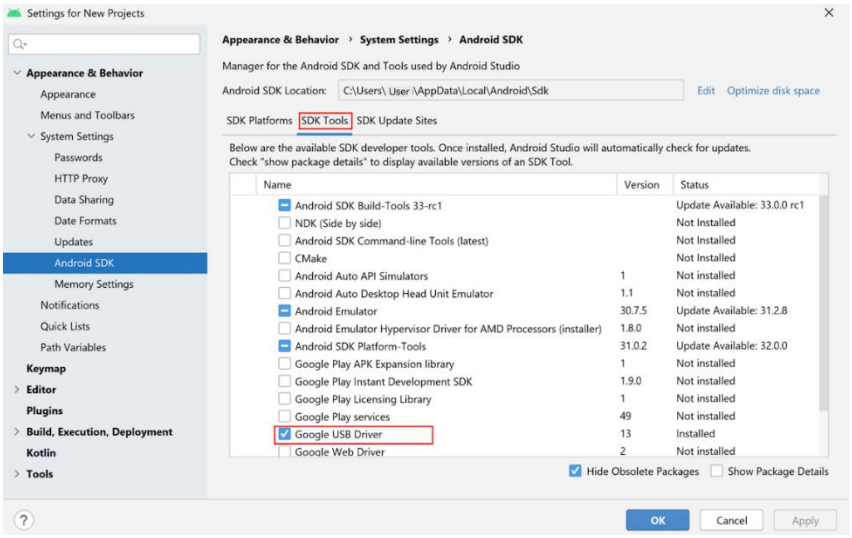
3. Zainstaluj sterownik Google USB (tylko Windows)

Jeśli zainstalowałeś Android Studio na komputerze z systemem Windows, musisz zainstalować sterownik urządzenia USB, zanim będzie można uruchomić aplikację na urządzeniu fizycznym.

Uwaga: w przypadku systemu Ubuntu Linux postępuj zgodnie z instrukcjami w dokumentacji Uruchamianie aplikacji na urządzeniu sprzętowym .

1. W Android Studio kliknij **Narzędzia > Menedżer SDK** .

Zostanie wyświetlone okno Ustawienia **dla nowych projektów .**



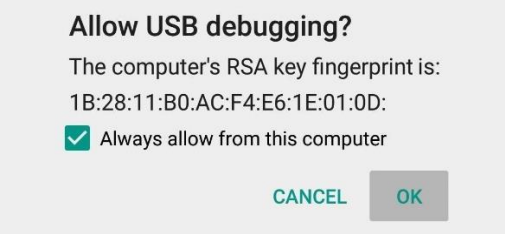
1. Kliknij kartę **Narzędzia SDK** .
2. Wybierz **Sterownik USB Google** i kliknij **OK** .

Po zakończeniu pliki sterownika są pobierane do `android_sdk/extras/google/usb_driver`katalogu. Teraz powinno być możliwe połączenie i uruchomienie aplikacji z Android Studio.

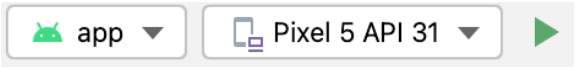
4. Uruchom aplikację na urządzeniu z systemem Android (wszystkie systemy operacyjne)

Teraz możesz podłączyć swoje urządzenie i uruchomić aplikację z Android Studio.

1. Połącz urządzenie z systemem Android z komputerem programistycznym za pomocą kabla USB. Na urządzeniu powinno pojawić się okno dialogowe z prośbą o zezwolenie na debugowanie USB.



1. Wybierz opcję **Zawsze zezwalaj** na zapamiętywanie tego komputera. Kliknij **OK** .
2. W Android Studio na komputerze upewnij się, że Twoje urządzenie jest wybrane z listy rozwijanej. Kliknij  .



1. Wybierz swoje urządzenie, a następnie kliknij **OK** .

Android Studio instaluje aplikację na Twoim urządzeniu i uruchamia ją.

Uwaga: W przypadku Android Studio 3.6 i nowszych urządzenie fizyczne jest wybierane automatycznie, gdy urządzenie jest połączone z włączonym debugowaniem.

Uwaga : jeśli na Twoim urządzeniu działa platforma Android, która nie jest zainstalowana w Android Studio, możesz zobaczyć komunikat z pytaniem, czy chcesz zainstalować wymaganą platformę. Kliknij **Zainstaluj i kontynuuj** , a następnie kliknij **Zakończ** po zakończeniu procesu.

5. Rozwiązywanie problemów

- Jeśli na komputerze działa system Linux lub Windows i nie możesz uruchomić aplikacji na fizycznym urządzeniu z systemem Android, zobacz Uruchamianie aplikacji na urządzeniu sprzętowym, aby uzyskać dodatkowe instrukcje.
- Jeśli na komputerze działa system Windows, a instalacja emulatora nie działa, zobacz Instalowanie sterowników USB OEM dla odpowiedniego sterownika USB dla urządzenia.
- Jeśli Android Studio nie rozpoznaje Twojego urządzenia, odłącz kabel USB i podłącz go ponownie. Uruchom ponownie Android Studio.
- Jeśli komputer nadal nie znajduje urządzenia lub deklaruje, że jest nieautoryzowane, odłącz kabel USB. Następnie na urządzeniu dotknij **Ustawienia > Opcje programisty > Odwołaj autoryzacje debugowania USB** . Ponownie podłącz urządzenie do komputera. Po wyświetleniu monitu udziel autoryzacji.

6. Wniosek

Dowiedziałeś się, jak uruchomić aplikację w Android Studio na fizycznym urządzeniu z Androidem!

7. Dowiedz się więcej

- Słownictwo dla Androida Podstawy w Kotlin
- Uruchamiaj aplikacje na urządzeniu sprzętowym
- Zainstaluj sterowniki USB OEM
- Pobierz dysk USB Google

Poznaj podstawy testów na Androida

1. Zanim zaczniesz

To laboratorium kodowania nauczy Cię o testowaniu i stosowaniu automatycznego testowania w aplikacjach na Androida.

Warunki wstępne

- Podstawowa wiedza o tym, jak poruszać się po katalogach projektów w Android Studio.

Czego się nauczysz

- Czym jest testowanie.
- Czym jest testowanie automatyczne.
- Czym są testy jednostkowe i oprzyrządowania.
- Gdzie znaleźć pliki testowe jednostki i oprzyrządowania w projekcie systemu Android.

Czego potrzebujesz

- Komputer z zainstalowanym Android Studio.
- Projekt utworzony w poprzednim ćwiczeniu z programowania w tej ścieżce.

2. Co to jest testowanie?

Testowanie w kontekście oprogramowania to ustrukturyzowana metoda sprawdzania oprogramowania, aby upewnić się, że działa poprawnie. Testowanie automatyczne to rzeczywisty kod, który sprawdza, czy inny napisany przez Ciebie fragment kodu działa poprawnie.

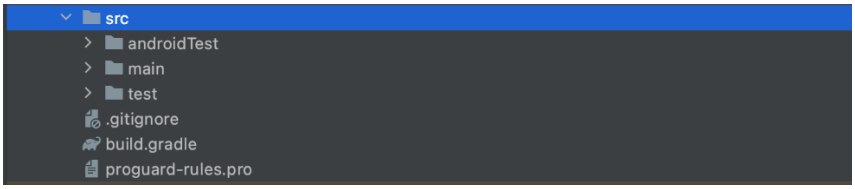
Testowanie oprogramowania jest korzystne, ponieważ pozwala wyeliminować błędy przed wypuszczeniem kodu na wolność; jest to niezbędne dla pozytywnego doświadczenia użytkownika.

Podczas gdy ręczne testowanie prawie zawsze ma swoje miejsce, testowanie w Androidzie często można zautomatyzować. W trakcie kursu Android Basics in Kotlin koncentrujesz się na automatycznych testach, aby przetestować kod aplikacji i wymagania funkcjonalne samej aplikacji. W tym laboratorium nauczysz się podstaw testowania w systemie Android. W późniejszych ćwiczeniach z programowania poznasz bardziej zaawansowane praktyki testowania aplikacji na Androida.

Gdy zapoznasz się z programowaniem i testowaniem aplikacji na Androida, powinieneś regularnie pisać testy wraz z kodem aplikacji. Tworzenie testu za każdym razem, gdy tworzysz nową funkcję w swojej aplikacji, zmniejsza obciążenie pracą w miarę rozwoju aplikacji. Zapewnia również wygodny sposób na upewnienie się, że aplikacja działa poprawnie, bez poświęcania zbyt wiele czasu na ręczne testowanie aplikacji.

3. Wprowadzenie do testów automatycznych

Test automatyczny to fragment kodu, który zapewnia, że inny napisany przez Ciebie fragment kodu działa poprawnie i nadal działa poprawnie w miarę rozwoju i zmian projektu. Zautomatyzowane testowanie jest istotną częścią całego procesu tworzenia oprogramowania, a tworzenie systemu Android nie jest wyjątkiem. W związku z tym nie ma lepszego czasu na wprowadzenie go niż teraz! Kiedy tworzyłeś swoją pierwszą aplikację na Androida, prawdopodobnie zauważyłeś, że główna aktywność znajduje się w podfolderze `main`katalogu. W `src`katalogu mogłeś również zauważyć katalogi `test`i . `androidTest`W tych dwóch katalogach pisany jest kod testowy. W rozwoju Androida istnieją dwa rodzaje testów automatycznych: testy jednostkowe i testy oprzyrządowania. Te dwa katalogi reprezentują te dwie kategorie testów.



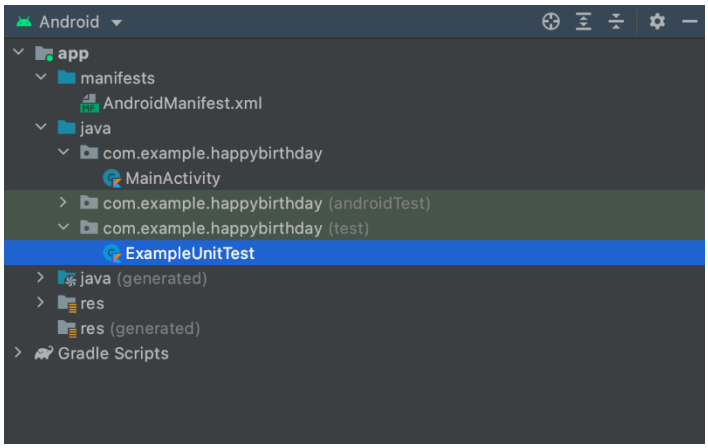
Znajdź kod testu jednostkowego

Testy lokalne w Androidzie znajdują się w `test`katalogu i zazwyczaj są to testy jednostkowe. Testy jednostkowe bezpośrednio testują mały fragment kodu, aby upewnić się, że działa poprawnie. Dzięki testom jednostkowym możesz testować funkcje, klasy i właściwości. Testy lokalne są wykonywane na wirtualnej maszynie Java, co oznacza, że działają w środowisku programistycznym bez konieczności korzystania z urządzenia lub emulatora. To fantazyjny sposób na powiedzenie, że testy jednostkowe są uruchamiane na twoim komputerze. Android Studio jest gotowy do automatycznego uruchamiania testów lokalnych.

Android Studio automatycznie generuje prosty test jednostkowy za każdym razem, gdy tworzysz nowy projekt. To samo dzieje się w przypadku testów oprzyrządowania. Należy zauważyć, że te testy tak naprawdę nie robią niczego istotnego. Służą tylko jako symbol zastępczy. Na razie zamierzasz tylko sprawdzić, gdzie znaleźć pliki testowe. Zagłębiasz się w treść tych wygenerowanych testów w późniejszej ścieżce.

Aby znaleźć kod testu jednostkowego:

1. Otwórz aplikację Kartka Urodzinowa z poprzedniego projektu.
2. W razie potrzeby wybierz **Android** z menu nawigacyjnego.
3. Kliknij **app** > **java** > **com.example.happybirthday (test)** > **ExampleUnitTest** .



Znajdź kod testu oprzyrządowania

W kontekście Android Development test oprzyrządowania jest terminem oznaczającym zwykle test interfejsu użytkownika (test interfejsu użytkownika). Testy oprzyrządowania umożliwiają testowanie części aplikacji, które zależą od cykli życia aktywności i fragmentów oraz interfejsów API i usług platformy.

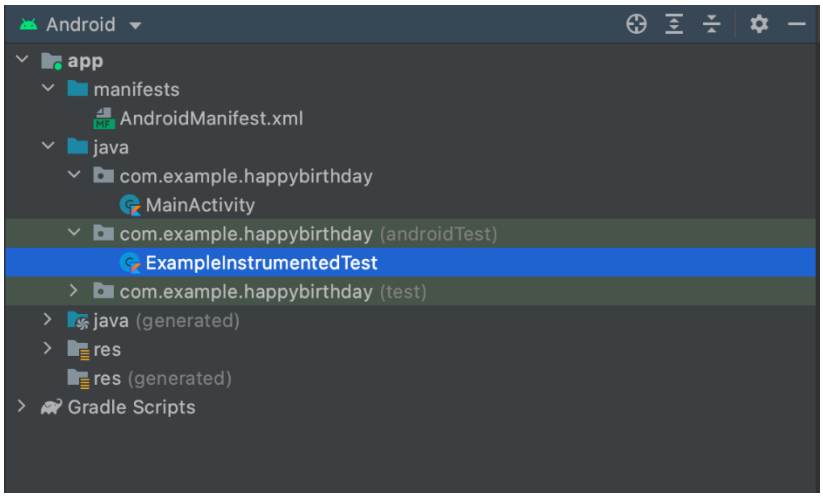
W przeciwieństwie do testów jednostkowych, testy interfejsu użytkownika nie testują kodu bezpośrednio. Zamiast tego testują interfejs użytkownika, aby upewnić się, że są wyświetlane prawidłowe składniki interfejsu użytkownika i że interfejs użytkownika zachowuje się zgodnie z oczekiwaniami podczas wykonywania akcji w interfejsie użytkownika. Inną różnicą jest to, że wszystkie testy oprzyrządowania muszą być uruchamiane na urządzeniu fizycznym lub emulatorze. We wcześniejszej ścieżce skonfigurowałeś emulator, więc ten krok jest już załatwiony.

Kiedy uruchamiasz test instrumentacji w systemie Android, tak naprawdę kod testowy jest wbudowany we własny pakiet APK, tak jak zwykła aplikacja na Androida. APK to skompresowany plik, który zawiera cały kod i pliki niezbędne do uruchomienia aplikacji na urządzeniu lub emulatorze. Ten testowy pakiet APK jest zainstalowany na urządzeniu lub emulatorze wraz ze zwykłym pakietem APK aplikacji. Następnie testowy pakiet APK uruchamia swoje testy względem pakietu APK aplikacji.

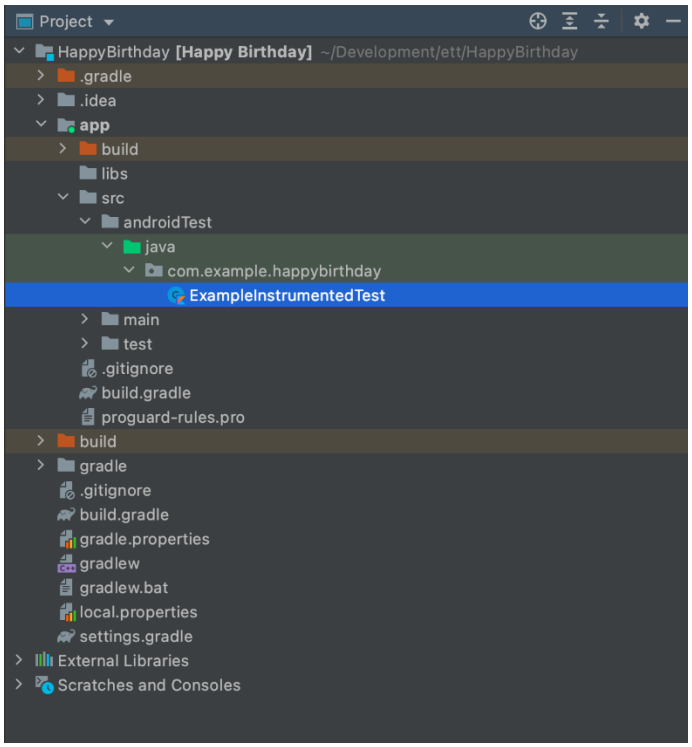
Zanim uruchomisz test, spójrz na to, co robi kod.

Aby znaleźć kod testu oprzyrządowania:

1. Jeśli jesteś w widoku projektu systemu **Android** , kliknij **app** > **java** > **com.example.happybirthday (androidTest)** > **ExampleInstrumentedTest** .



1. Jeśli jesteś w widoku **projektu** projektu,
   kliknij **HappyBirthday** > **app** > **src** > **androidTest** > **java** > **com.example.happybirthday** > **ExampleInstrumentedTest** .



4. Gratulacje

Dowiedziałeś się, czym jest testowanie w systemie Android i jak znaleźć testy jednostkowe i oprzyrządowanie w systemie Android.

Ścieżka 3. Zbuduj podstawowy układ

Dowiedz się, jak dodawać obrazy i tekst do aplikacji na Androida.

Utwórz aplikację Kartka Urodzinowa

1. Wstęp

Podczas tego ćwiczenia z programowania zbudujesz prostą aplikację na Androida wyświetlającą tekst. Będziesz mógł pozycjonować tekst na ekranie dzięki lepszemu poznaniu komponentów interfejsu użytkownika (UI) w systemie Android.

Warunki wstępne

- Jak stworzyć nową aplikację w Android Studio.
- Jak uruchomić aplikację w emulatorze lub na urządzeniu z Androidem.

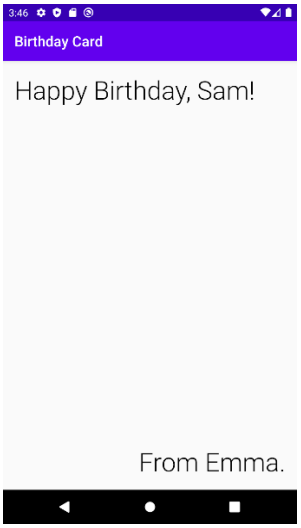
Czego się nauczysz

- Czym są elementy interfejsu użytkownika, takie jak `Views`i `ViewGroups`.
- Jak wyświetlić tekst w `TextView`aplikacji.
- Jak ustawić atrybuty, takie jak tekst, czcionka i margines na `TextView`.

Co zbudujesz

- Aplikacja na Androida wyświetlająca życzenia urodzinowe w formacie tekstowym.

Tak będzie wyglądać Twoja aplikacja, gdy skończysz.



Czego potrzebujesz

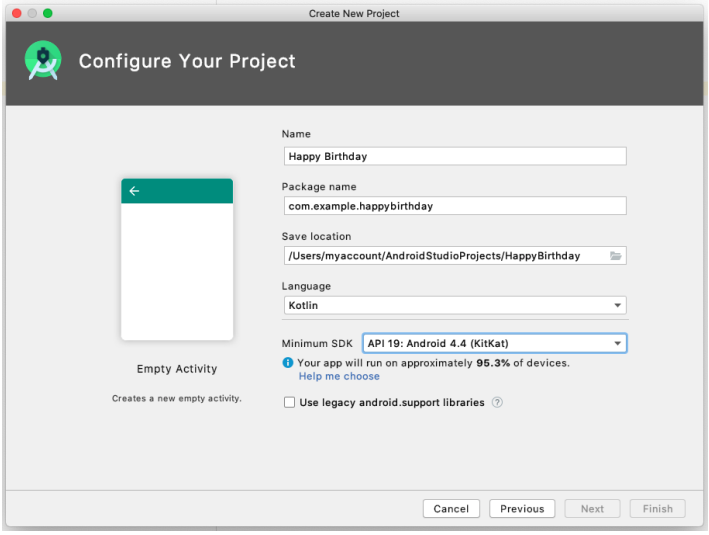
- Komputer z zainstalowanym Android Studio.

2. Skonfiguruj aplikację Happy Birthday

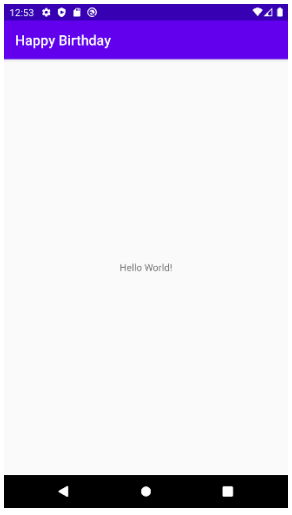
Utwórz projekt pustej aktywności

1. Aby rozpocząć, utwórz nowy projekt Kotlin w Android Studio, korzystając z szablonu **Empty Activity .**
2. Zadzwoń do aplikacji „Happy Birthday” z minimalnym poziomem interfejsu API 19 (KitKat).

Ważne: Jeśli nie wiesz, jak tworzyć nowy projekt w Android Studio, zobacz Tworzenie i uruchamianie pierwszej aplikacji na Androida , aby uzyskać szczegółowe informacje.



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



Po utworzeniu tej aplikacji Happy Birthday za pomocą szablonu Empty Activity, Android Studio skonfiguruje zasoby dla podstawowej aplikacji na Androida, w tym „Hello World!” wiadomość na środku ekranu. Podczas tego ćwiczenia kodowania dowiesz się, w jaki sposób ta wiadomość się tam dostaje, jak zmienić jej tekst, aby był bardziej przypominający życzenia urodzinowe oraz jak dodawać i formatować dodatkowe wiadomości.

O interfejsach użytkownika

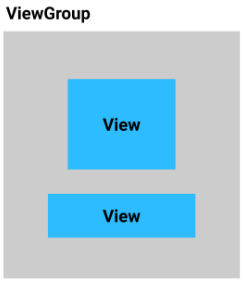
Interfejs użytkownika (UI) aplikacji to to, co widzisz na ekranie: tekst, obrazy, przyciski i wiele innych typów elementów. Jest to sposób, w jaki aplikacja pokazuje użytkownikowi rzeczy i jak użytkownik wchodzi w interakcję z aplikacją.

Każdy z tych elementów nazywa się `View`. Prawie wszystko, co widzisz na ekranie swojej aplikacji, to `View`. `Views`może być interaktywny, jak klikalny przycisk lub edytowalne pole wejściowe.

W tym ćwiczeniu z programowania będziesz pracować z rodzajem `View`, który służy do wyświetlania tekstu i nazywa się a `TextView`.

W `Views`aplikacji na Androida nie tylko same unoszą się na ekranie. `Views`mieć ze sobą relacje. Na przykład obraz może znajdować się obok tekstu, a przyciski mogą tworzyć rząd. Aby uporządkować `Views`, umieszczasz je w pojemniku. A `ViewGroup`jest pojemnikiem, w którym `View`mogą siedzieć przedmioty i odpowiada za uporządkowanie jego `Views`wnętrza. Układ lub *układ* , może się zmieniać w zależności od rozmiaru i proporcji ekranu urządzenia z Androidem, na którym działa aplikacja, a układ można dostosować do tego, czy urządzenie jest w trybie pionowym, czy poziomym.

Jednym z rodzajów `ViewGroup`jest `ConstraintLayout`, który pozwala `Views`elastycznie zaaranżować jego wnętrze.



O edytorze układu

Tworzenie interfejsu użytkownika poprzez aranżowanie `Views`i `ViewGroups`jest dużą częścią tworzenia aplikacji na Androida. Android Studio udostępnia narzędzie, które w tym pomoże, zwane **Edytorem układu** . Użyjesz **Edytora układu** , aby zmienić „Hello World!” wpisz tekst „Wszystkiego najlepszego!”, a później, aby nadać mu styl.

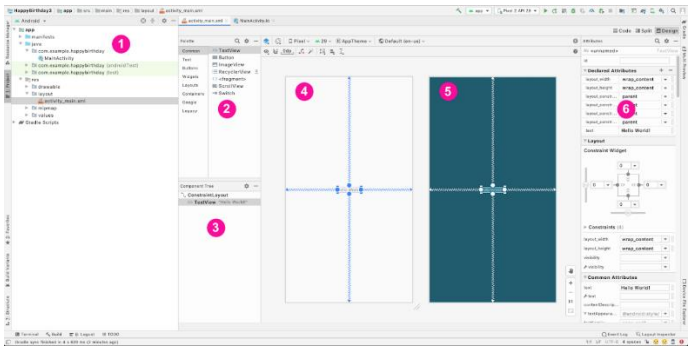
Po otwarciu **Edytor układu** zawiera wiele części. Większość z nich nauczysz się korzystać z tego ćwiczenia z programowania. Użyj zrzutu ekranu z adnotacjami poniżej, aby ułatwić rozpoznawanie okien w **Edytorze układu** . Dowiesz się więcej o każdej części podczas wprowadzania zmian w swojej aplikacji.

- Po lewej (1) znajduje się okno **projektu** , które widziałeś wcześniej. Zawiera listę wszystkich plików, które składają się na Twój projekt.
- W środku możesz zobaczyć dwa rysunki (4) i (5) przedstawiające układ ekranu Twojej aplikacji. Reprezentacja po lewej stronie (4) jest bliskim przybliżeniem tego, jak będzie wyglądał Twój ekran po uruchomieniu aplikacji. Nazywa się to widokiem **projektu** .
- Reprezentacja po prawej (5) to widok **schematu** , który może być przydatny w przypadku określonych operacji.
- Paleta **(** 2) zawiera listy różnych typów, `Views`które możesz dodać do swojej aplikacji.

- Drzewo **komponentów** (3) to inna reprezentacja widoków twojego ekranu. Zawiera listę wszystkich widoków ekranu.
- Po prawej stronie (6) znajdują się **Atrybuty** . Pokazuje atrybuty `View`i pozwala je zmienić.

Przeczytaj więcej o **Edytorze układu** i sposobie jego konfiguracji w przewodniku dla programistów .

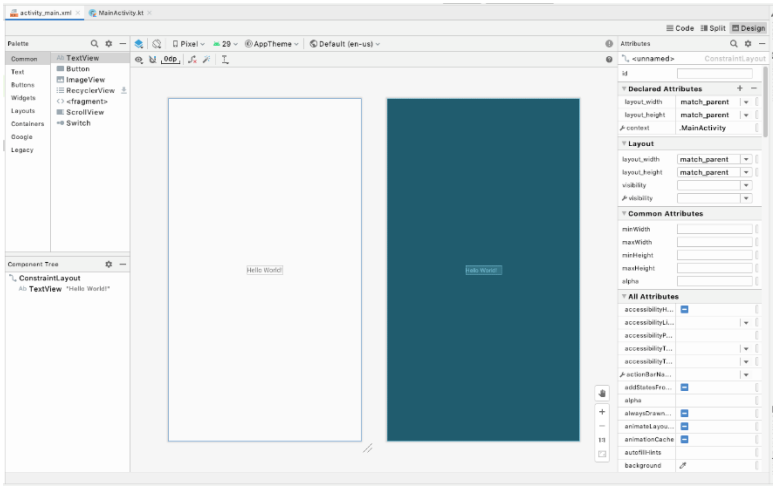
Zrzut ekranu z adnotacjami całego **edytora układu** :



Zróbmy kilka zmian w **Edytorze układu** , aby Twoja aplikacja bardziej przypominała kartkę urodzinową!

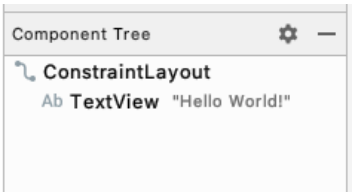
Zmień wiadomość Hello World

1. W Android Studio znajdź okno **Project** po lewej stronie.
2. Zwróć uwagę na te foldery i pliki: folder **aplikacji** zawiera większość plików aplikacji, które chcesz zmienić. **Folder res** jest przeznaczony na zasoby, takie jak obrazy lub układy ekranu. **Folder układu** jest przeznaczony na układy ekranu. Plik `activity_main.xml`zawiera opis układu ekranu.
3. Rozwiń folder **aplikacji** , następnie folder **res** , a następnie folder **układu** .
4. Kliknij dwukrotnie na `activity_main.xml`. Otwiera się to `activity_main.xml`w **Edytorze układu** i pokazuje układ, który opisuje w widoku **Projekt** .

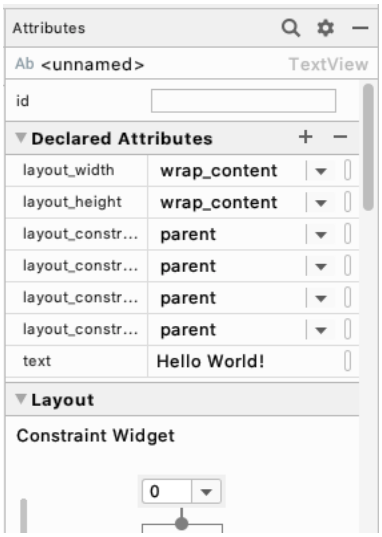


Uwaga: w tych ćwiczeniach z programowania często będziesz proszony o otwarcie pliku, tak jak w poprzednich krokach. W skrócie może to być skrót:
Otwórz **activity_main.xml** (**res** > **layout** > **activity_main.xml**) zamiast wymieniać każdy krok osobno.

1. Spójrz na listę widoków w **drzewie komponentów** . Zwróć uwagę, że pod nim znajduje się `ConstraintLayout`, i `TextView`. Reprezentują one interfejs Twojej aplikacji. Jest `TextView`wcięty, ponieważ znajduje się wewnątrz `ConstraintLayout`. W miarę dodawania kolejnych `Views`do `ConstraintLayout`, będą one dodawane do tej listy.
2. Zauważ, że `TextView`ma **"Hello World!"** obok, czyli tekst, który widzisz po uruchomieniu aplikacji.

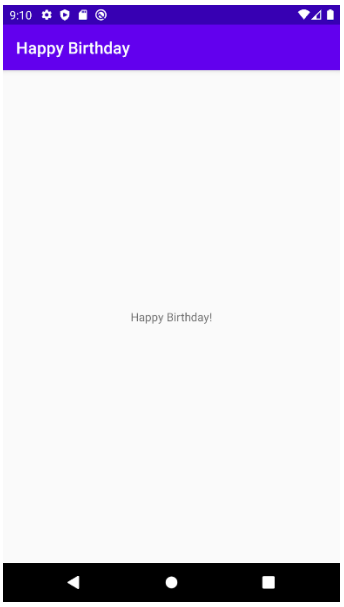


1. W **drzewie komponentów** kliknij `TextView`.
2. Znajdź **atrybuty** po prawej stronie.
3. Znajdź sekcję **Deklarowane atrybuty** .
4. Zauważ, że atrybut **text w sekcji Declared Attributes** zawiera **Hello World!** .



Atrybut **text** pokazuje tekst, który jest drukowany wewnątrz `TextView`.

1. Kliknij atrybut **tekstowy , w którym znajduje się Hello World!** tekst jest.
2. Zmień to na „ **Wszystkiego najlepszego!** , a następnie naciśnij klawisz **Enter** . Jeśli zobaczysz ostrzeżenie o zakodowanym na stałe ciągu, nie martw się tym na razie. Dowiesz się, jak pozbyć się tego ostrzeżenia w następnym ćwiczeniu z programowania.
3. Zauważ, że tekst zmienił się w **Widoku Projektu** ....(to fajne, możesz od razu zobaczyć swoje zmiany!)
4. Uruchom swoją aplikację, a teraz jest napisane **Happy Birthday!**



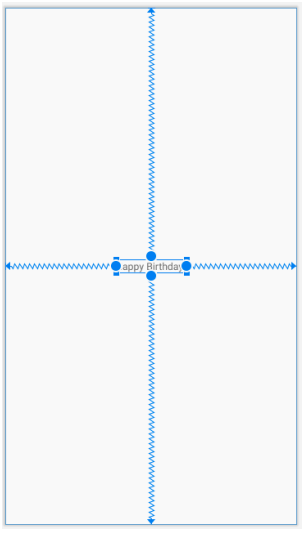
Dobra robota! Wprowadziłeś pierwsze zmiany w aplikacji na Androida.

3. Dodaj TextViews do układu

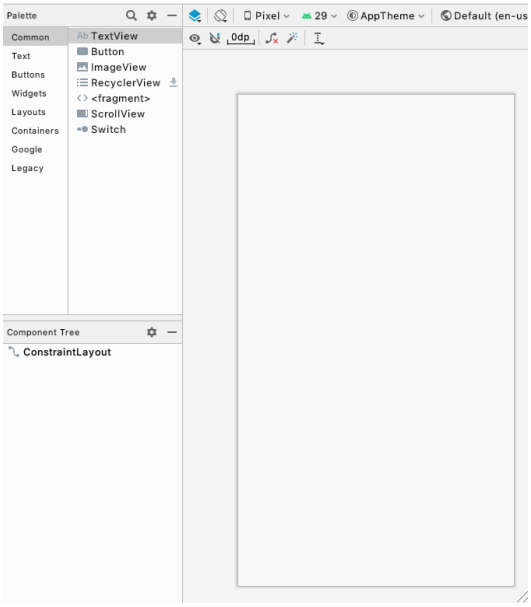
Kartka urodzinowa, którą tworzysz, wygląda inaczej niż ta, którą masz teraz w swojej aplikacji. Zamiast małego tekstu na środku potrzebujesz dwóch większych wiadomości, jednej w lewym górnym rogu i jednej w prawym dolnym rogu. W tym zadaniu usuniesz istniejące `TextView`, dodasz dwa nowe `TextViews`i nauczysz się umieszczać je w `ConstraintLayout`.

Usuń bieżący widok tekstu

1. W **Edytorze układu** kliknij, aby wybrać `TextView`środek układu.



1. Naciśnij klawisz **Usuń** . Android Studio usuwa `TextView`, a Twoja aplikacja wyświetla teraz tylko `ConstraintLayout`w **Edytorze układu** i **Drzewie komponentów** .

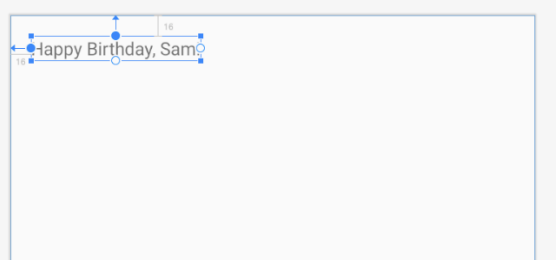


Porada: Jeśli chcesz powiększyć układ, możesz użyć elementów sterujących w prawym dolnym rogu **Edytora układu** , aby dostosować rozmiar. Kliknij najniższą ikonę, aby

powrócić do poziomu powiększenia, w którym cały układ mieści się na ekranie.

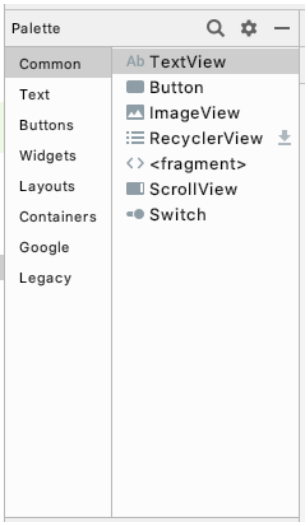
Dodaj widok tekstu

W tym kroku dodasz znak `TextView`w lewym górnym rogu swojej aplikacji, aby pomieścić życzenia urodzinowe.

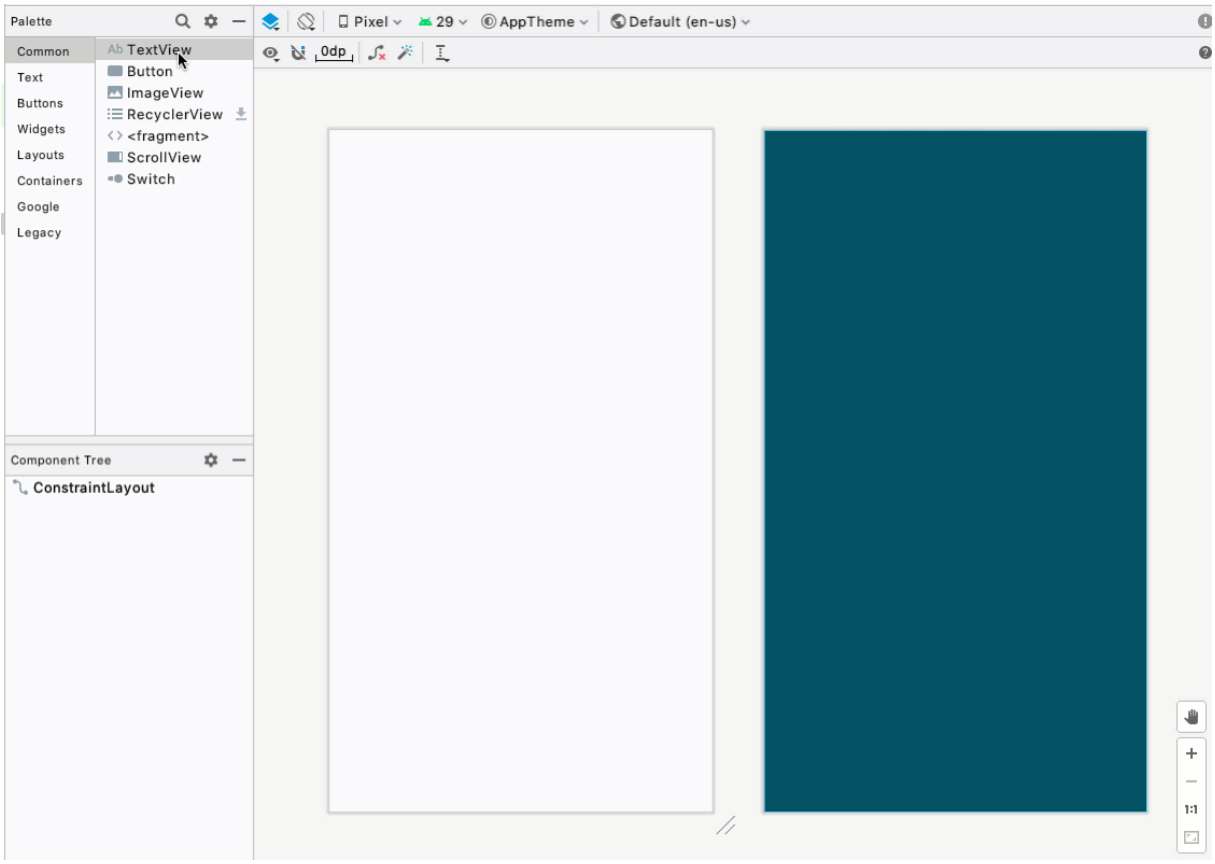


Paleta w **lewym** górnym rogu **Edytora układu** zawiera listy różnych typów `Views`, uporządkowane według kategorii, które możesz dodać do swojej aplikacji.

1. Znajdź `TextView`. Pojawia się zarówno w kategorii **Wspólne** , jak i **Tekst** .



1. Przeciągnij a `TextView`z **palety** w lewym górnym rogu powierzchni projektowej w **Edytorze układu** i upuść. Nie musisz być dokładny, po prostu upuść go w lewym górnym rogu.

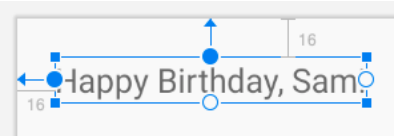


1. Zwróć uwagę na `TextView`dodanie i zwróć uwagę na czerwony wykrzyknik w **drzewie komponentów** .
2. Umieść wskaźnik myszy nad wykrzyknikiem, a zobaczysz komunikat ostrzegawczy, że widok nie jest ograniczony i przeskoczy do innej pozycji po uruchomieniu aplikacji. Naprawisz to w następnym kroku.

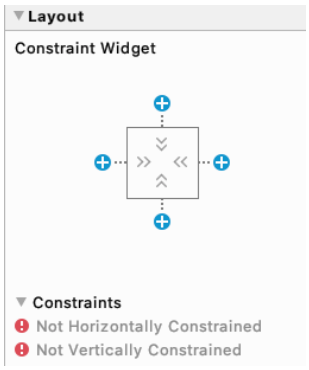
Ustaw widok tekstu

W przypadku kartki urodzinowej `TextView`musi znajdować się w lewym górnym rogu z pewną przestrzenią wokół niej. Aby naprawić ostrzeżenie, dodasz ograniczenia do `TextView`, które poinstruują Twoją aplikację, jak ją ustawić. Ograniczenia to wskazówki i ograniczenia określające, gdzie w układzie `View`może znajdować się a.

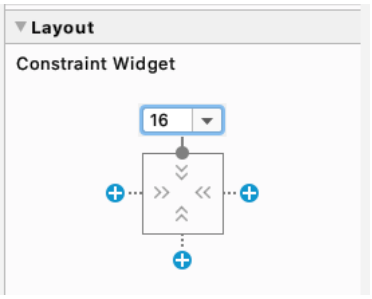
Ograniczenia, które dodasz na górze i na lewo, będą miały marginesy. Margines określa odległość a `View`od krawędzi pojemnika.



1. W **Atrybutach** po prawej stronie znajdź **Widżet Ograniczenie** w sekcji **Układ** . Kwadrat reprezentuje Twój pogląd.



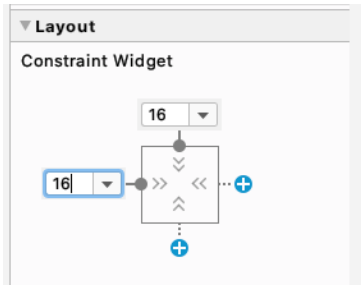
1. Kliknij **+** u góry kwadratu. Jest to ograniczenie między górną częścią widoku tekstu a górną krawędzią układu ograniczenia.
2. Pojawi się pole z numerem do ustawienia górnego marginesu. Margines to odległość od `TextView`krawędzi kontenera do krawędzi `ConstraintLayout`. Widoczna liczba będzie się różnić w zależności od tego, gdzie upuściłeś `TextView`. Kiedy ustawisz górny margines, Android Studio automatycznie dodaje również ograniczenie od góry widoku tekstu do góry `ConstrainLayout`.



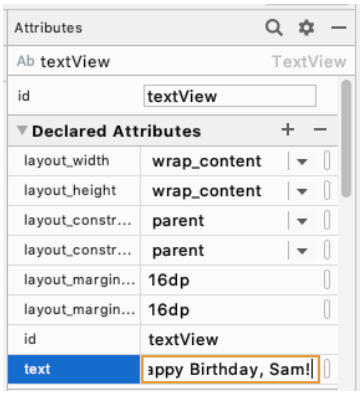
1. Zmień górny margines na 16.

Uwaga: Jednostką marginesów i innych odległości w interfejsie użytkownika są *piksele niezależne od gęstości* (dp). To jak centymetry lub cale, ale dla odległości na ekranie. Android tłumaczy tę wartość na odpowiednią liczbę rzeczywistych pikseli dla każdego urządzenia. Jako podstawa 1dp wynosi około 1/160 cala, ale w przypadku niektórych urządzeń może być większe lub mniejsze.

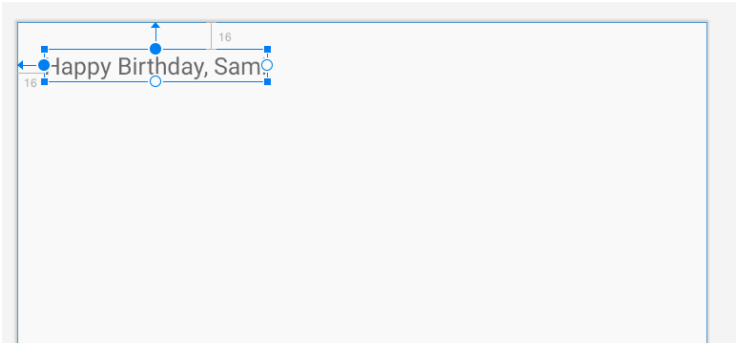
1. Zrób to samo dla lewego marginesu.



1. Ustaw **tekst** , aby życzyć przyjacielowi wszystkiego najlepszego, na przykład „Happy Birthday, Sam!” i naciśnij **Enter** .



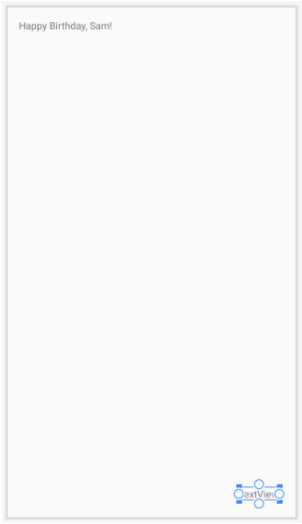
1. Zwróć uwagę, że widok **Projekt** zostanie zaktualizowany, aby pokazać, jak będzie wyglądać Twoja aplikacja.



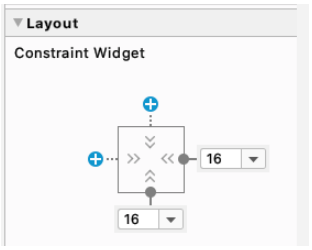
Dodaj i umieść kolejny TextView

Twoja kartka urodzinowa potrzebuje drugiego wiersza tekstu w prawym dolnym rogu, który dodasz w tym kroku w taki sam sposób, jak w poprzednim zadaniu. Jak myślisz, jakie `TextView`powinny być marginesy?

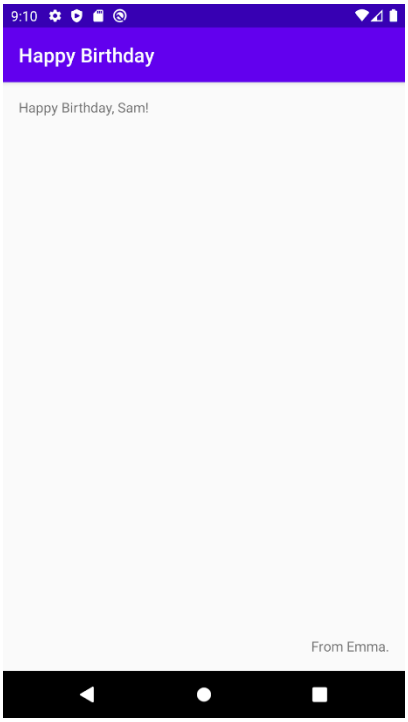
1. Przeciągnij nowy `TextView`z **palety** i upuść go w prawym dolnym rogu widoku aplikacji w Edytorze układu.



1. Ustaw prawy margines na 16.
2. Ustaw dolny margines na 16.



1. W **Atrybutach** ustaw atrybut **tekstowy** do podpisywania karty, na przykład „Od Emmy”.
2. Uruchom swoją aplikację. Powinieneś zobaczyć swoje życzenia urodzinowe w lewym górnym rogu i swój podpis w prawym dolnym rogu.

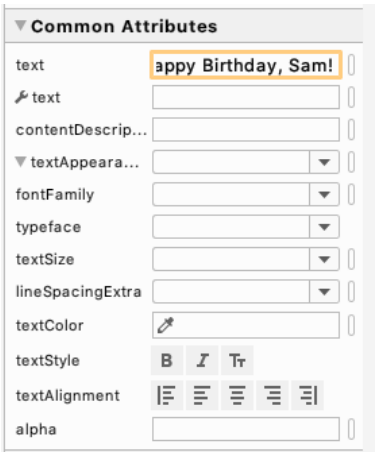


Gratulacje! Dodałeś i umieściłeś w swojej aplikacji niektóre elementy interfejsu użytkownika.

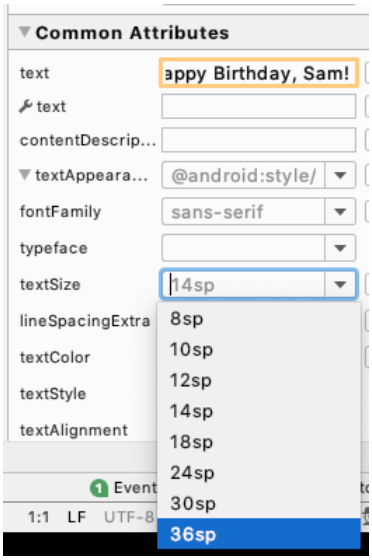
4. Dodaj styl do tekstu

Dodałeś tekst do interfejsu użytkownika, ale nie wygląda on jeszcze jak ostateczna aplikacja. W tym zadaniu dowiesz się, jak zmienić rozmiar, kolor tekstu i inne atrybuty, które wpływają na wygląd `TextView`. Możesz także eksperymentować z różnymi czcionkami.

1. Kliknij pierwszy `TextView`w **drzewie komponentów** i znajdź sekcję **Common Attributes okna Attributes** . Może być konieczne przewinięcie w dół, aby go znaleźć.
2. Zwróć uwagę na różne atrybuty, w tym **fontFamily** , **textSize** i **textColor** .



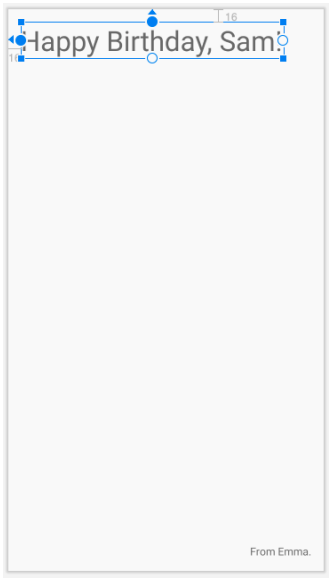
1. Poszukaj **tekstuWygląd** .
2. Jeśli **textAppearance** nie jest rozwinięty, kliknij trójkąt w dół.
3. Ustaw **textSize** na **36sp** .



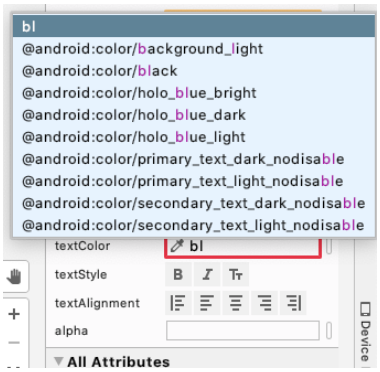
Uwaga: Tak jak dp jest jednostką miary odległości na ekranie, **sp** jest jednostką miary rozmiaru czcionki. Elementy interfejsu użytkownika w aplikacjach na Androida używają dwóch różnych jednostek miary, *pikseli niezależnych od gęstości* (**dp**), których użyłeś wcześniej do układu, oraz *pikseli skalowalnych* (**sp**), które są używane podczas ustawiania

rozmiaru tekstu. Domyślnie sp ma taki sam rozmiar jak dp, ale jego rozmiar zmienia się w zależności od preferowanego rozmiaru tekstu użytkownika.

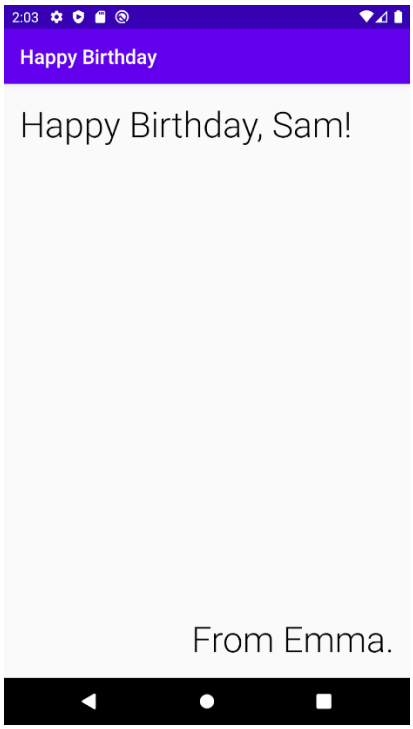
1. Zwróć uwagę na zmianę w **Edytorze układu** .



1. Zmień **fontFamily** na **casual** .
2. Wypróbuj różne czcionki, aby zobaczyć, jak wyglądają. Istnieje jeszcze więcej możliwości wyboru czcionek na dole listy, w sekcji **Więcej czcionek...**
3. Po zakończeniu próbowania różnych czcionek ustaw **fontFamily** na **sans-serif-light** .
4. Kliknij pole edycji atrybutu **textColor** i zacznij pisać **black** . Zwróć uwagę, że podczas pisania Android Studio wyświetla listę kolorów zawierających wpisany do tej pory tekst.



1. Wybierz **@android:color/black** z listy kolorów i naciśnij **Enter** .
2. W polu `TextView`z podpisem zmień wartości **textSize** , **textColor** i **fontFamily tak** , aby były zgodne.
3. Uruchom swoją aplikację i spójrz na wynik.



Gratulacje, wykonałeś pierwsze kroki w tworzeniu aplikacji kartki urodzinowej!

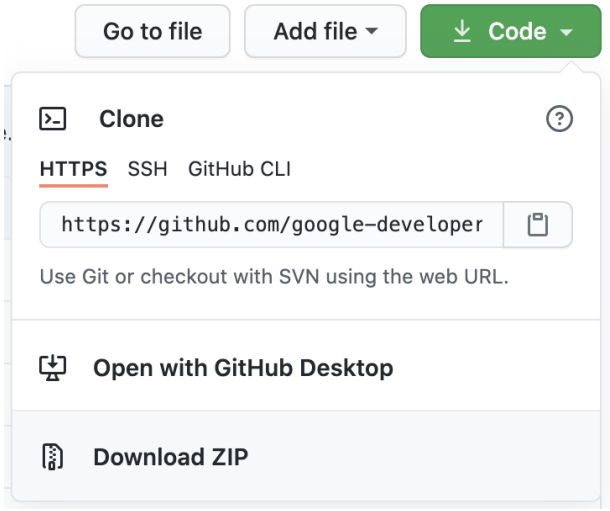
5. Rozwiązanie

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

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

Pobierz kod

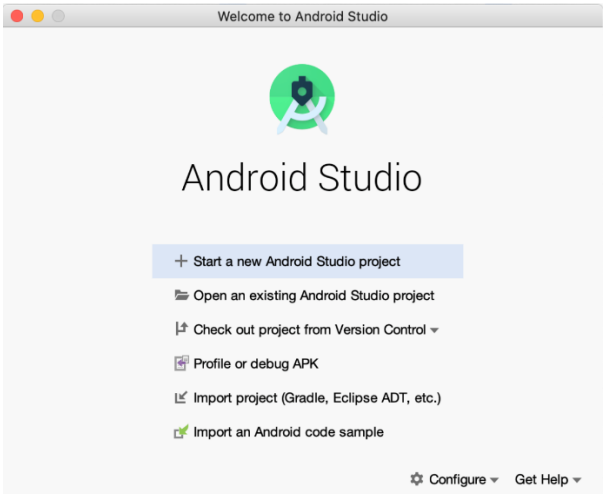
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



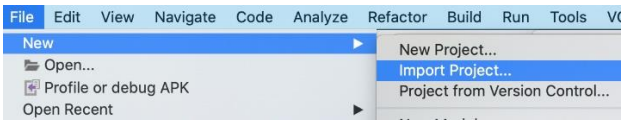
1. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
2. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
3. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego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



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

Alert: w Android Studio, jeśli zobaczysz błąd „ **Wykryto platformę Android. Kliknij, aby skonfigurować** ”, projekt nie jest wykrywany lub przycisk uruchamiania jest wyłączony, upewnij się, że otwierasz podkatalog **HappyBirthday** w Android Studio, a nie katalog nadrzędny.

6. Podsumowanie

- Edytor **układu** pomaga w tworzeniu interfejsu użytkownika aplikacji na Androida.

- Prawie wszystko, co widzisz na ekranie swojej aplikacji, to `View`.
- A `TextView`to element interfejsu użytkownika do wyświetlania tekstu w Twojej aplikacji.
- A `ConstraintLayout`to kontener na inne elementy interfejsu użytkownika.
- `Views`muszą być ograniczone poziomo i pionowo w ramach `ConstraintLayout`.
- Jednym ze sposobów na ustawienie a `View`jest margines.
- Margines mówi, jak daleko a `View`jest od krawędzi pojemnika, w którym się znajduje.
- Możesz ustawić atrybuty na `TextView`czcionkę, rozmiar tekstu i kolor.

7. Dowiedz się więcej

- Słownictwo dla Androida Podstawy w Kotlin
- `View`
- `TextView`
- `ConstraintLayout`
- dp vs. sp
- Edytor układu w Android Studio

Dodaj obrazy do aplikacji na Androida

1. Wstęp

W tym ćwiczeniu z programowania dowiesz się, jak dodawać obrazy do aplikacji za pomocą `ImageView`.

Warunki wstępne

- Jak stworzyć i uruchomić nową aplikację w Android Studio.
- Jak dodawać i usuwać oraz ustawiać atrybuty za `TextViews`pomocą Edytora układu.

Czego się nauczysz

- Jak dodać obraz lub zdjęcie do aplikacji na Androida.
- Jak wyświetlić obraz w aplikacji za pomocą `ImageView`.
- Jak wyodrębnić tekst do zasobu ciągu, aby ułatwić tłumaczenie aplikacji i ponowne użycie ciągów.
- Jak sprawić, by Twoja aplikacja była dostępna dla jak największej liczby osób.

Co zbudujesz

- Rozszerz aplikację Happy Birthday, aby dodać obraz.

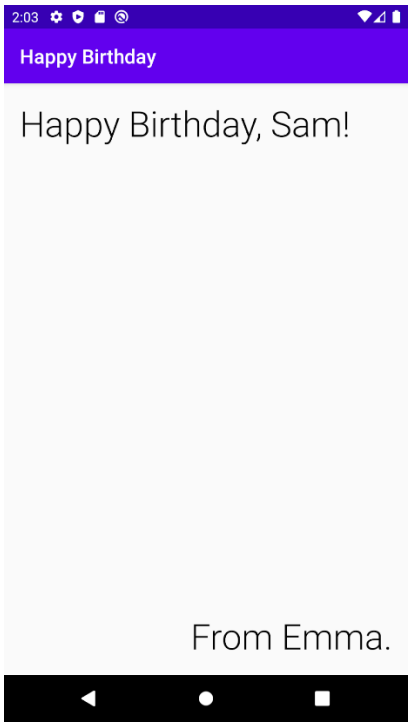
Czego potrzebujesz

- Komputer z zainstalowanym Android Studio.

- Aplikacja z ćwiczenia kodowania Utwórz kartę urodzinową.

2. Skonfiguruj swoją aplikację

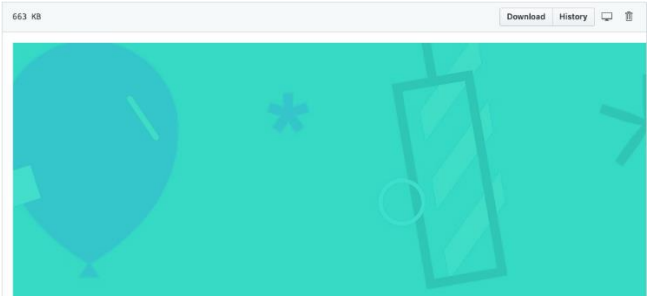
1. Otwórz projekt z poprzedniego ćwiczenia z programowania w Android Studio. Możesz użyć kodu rozwiązania lub utworzonego kodu. Po uruchomieniu aplikacja powinna wyglądać tak.



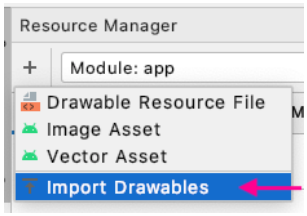
Dodaj obraz do swojego projektu

W tym zadaniu pobierzesz obraz z Internetu i dodasz go do swojej aplikacji Happy Birthday.

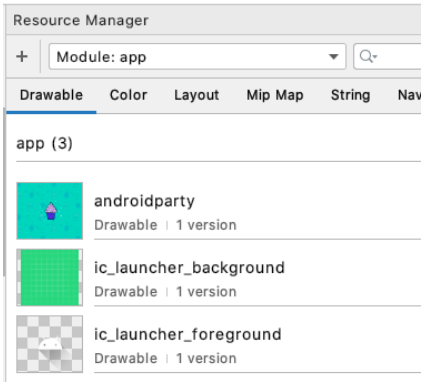
1. Kliknij tutaj, aby uzyskać dostęp do obrazu swojej kartki urodzinowej na Github.
2. Kliknij przycisk **Pobierz** po prawej stronie. To wyświetla obraz sam.



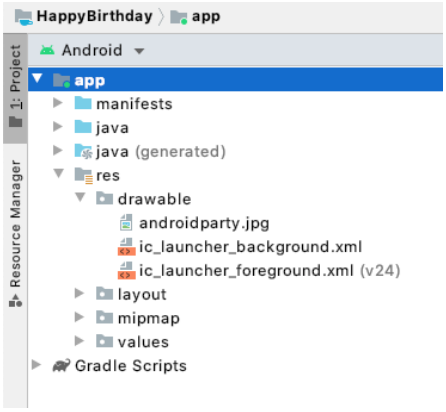
1. Kliknij obraz prawym przyciskiem myszy i zapisz plik na komputerze jako **androidparty.png** . Zanotuj, gdzie go zapisałeś (na przykład folder **Pobrane).**
2. W Android Studio kliknij menu **Widok > Narzędzia Windows > Menedżer zasobów lub kliknij kartę Menedżer zasobów** po lewej stronie okna **projektu** .
3. Kliknij przycisk **+** poniżej **Menedżera zasobów** i wybierz opcję **Importuj elementy do rysowania** . Spowoduje to otwarcie przeglądarki plików.



1. W przeglądarce plików znajdź pobrany plik obrazu i kliknij **Otwórz** .
2. Kliknij **Dalej** . Android Studio wyświetla podgląd obrazu.
3. Kliknij **Importuj** .
4. Jeśli obraz został pomyślnie zaimportowany, Android Studio doda obraz do listy do **rysowania** . Ta lista zawiera wszystkie Twoje obrazy i ikony aplikacji. Możesz teraz użyć tego obrazu w swojej aplikacji.



1. Przełącz się z powrotem do widoku projektu, klikając opcję **Widok > Okna narzędzi > Projekt** w menu lub kartę **Projekt** po lewej stronie.
2. Potwierdź, że obraz znajduje się w folderze do **rysowania** aplikacji, rozwijając **app > res > drawable** .



3. Dodaj widok obrazu

Aby wyświetlić obraz w Twojej aplikacji, potrzebuje miejsca do wyświetlenia. Tak jak używasz a `TextView`do wyświetlania tekstu, możesz użyć a `ImageView`do wyświetlania obrazów.

W tym zadaniu dodasz ikonę ImageViewdo swojej aplikacji i ustawisz jej obraz na pobrany obraz babeczki. Następnie ustawisz go i dostosujesz jego rozmiar tak, aby wypełniał ekran.

Dodaj ImageView i ustaw jego obraz

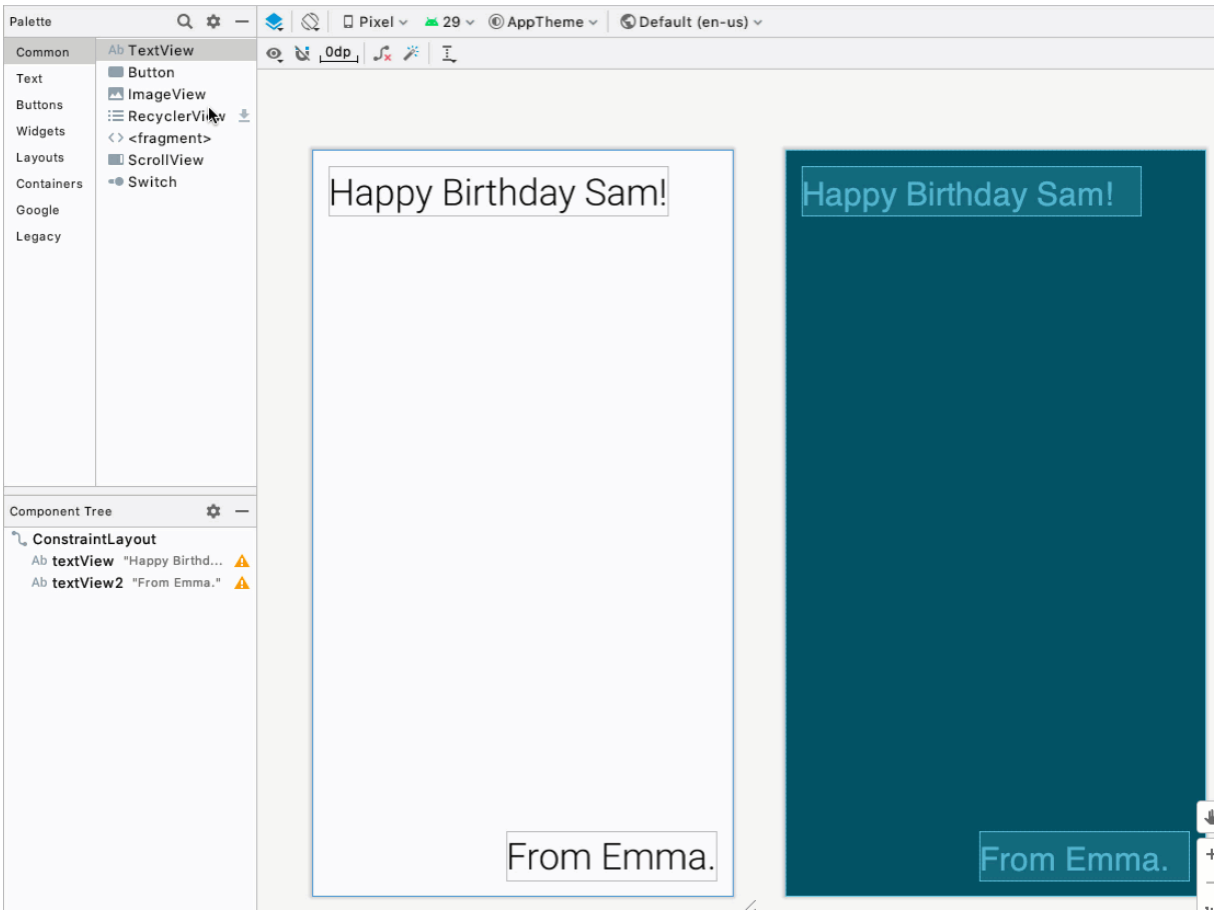
1. W oknie **projektu** otwórz **activity_main.xml** (**app > res > layout > activity_main.xml**).

Wskazówka: jeśli nie widzisz **Edytora układu** , kliknij przycisk Tryb **projektowania w prawym górnym rogu.**

1. W **Edytorze układu** przejdź do **palety** i przeciągnij ImageViewdo swojej aplikacji. Upuść go blisko środka i nie nakładaj się na żaden tekst

Otworzy się okno dialogowe **Wybierz zasób .** To okno dialogowe zawiera listę wszystkich zasobów graficznych dostępnych w Twojej aplikacji. Zwróć uwagę na obraz urodzin znajdujący się pod **zakładką** Do rysowania . *Zasób* do rysowania to ogólna koncepcja grafiki, którą można narysować na ekranie. Zawiera obrazy, mapy bitowe i ikony oraz wiele innych rodzajów rysowanych zasobów.

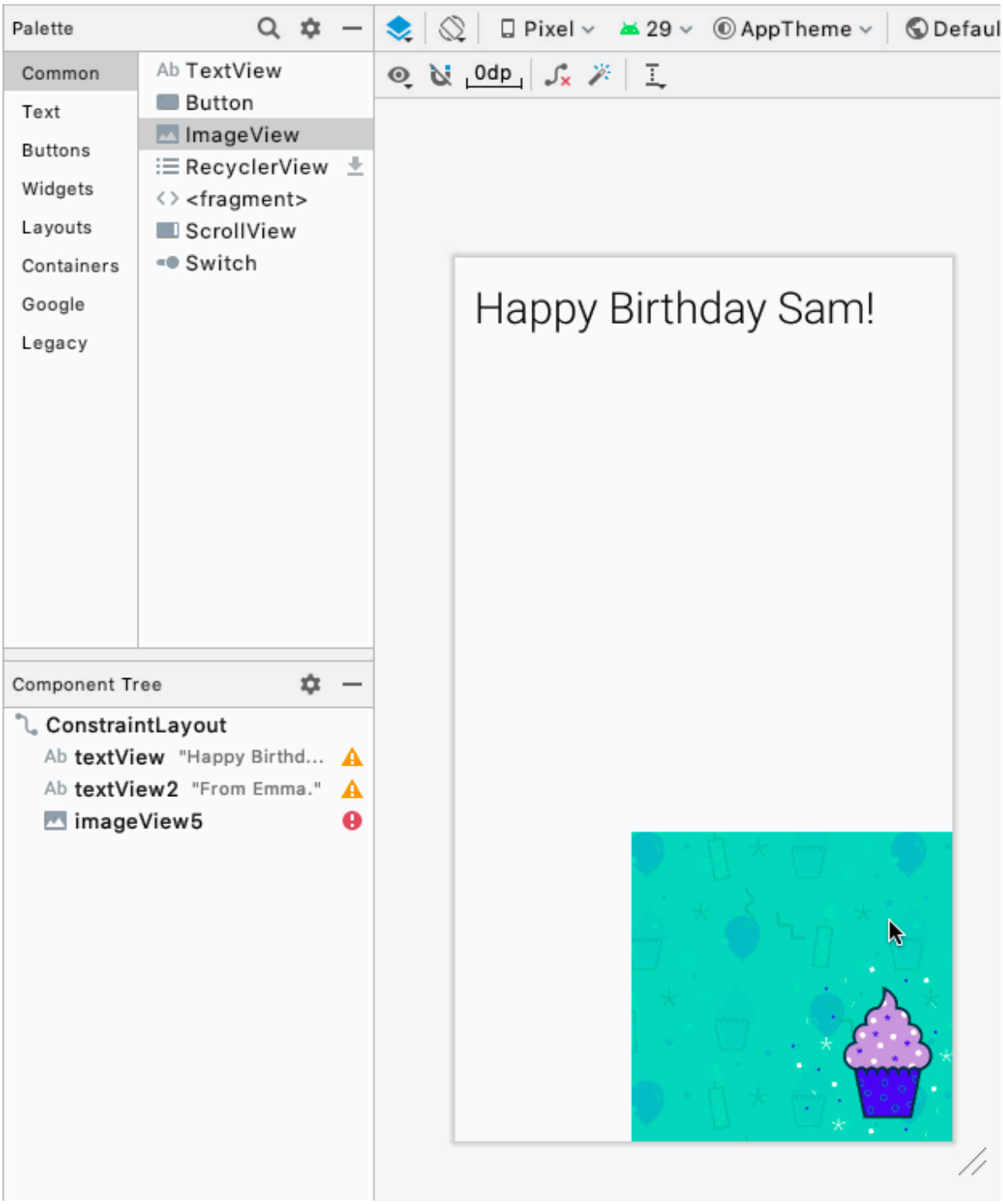
1. W oknie dialogowym **Wybierz zasób** znajdź obraz ciasta na liście do **rysowania** .
2. Kliknij na obrazek, a następnie kliknij **OK** .



Spowoduje to dodanie obrazu do Twojej aplikacji, ale prawdopodobnie nie jest we właściwym miejscu i nie wypełnia ekranu. Naprawisz to w następnym kroku.

Pozycja i rozmiar ImageView

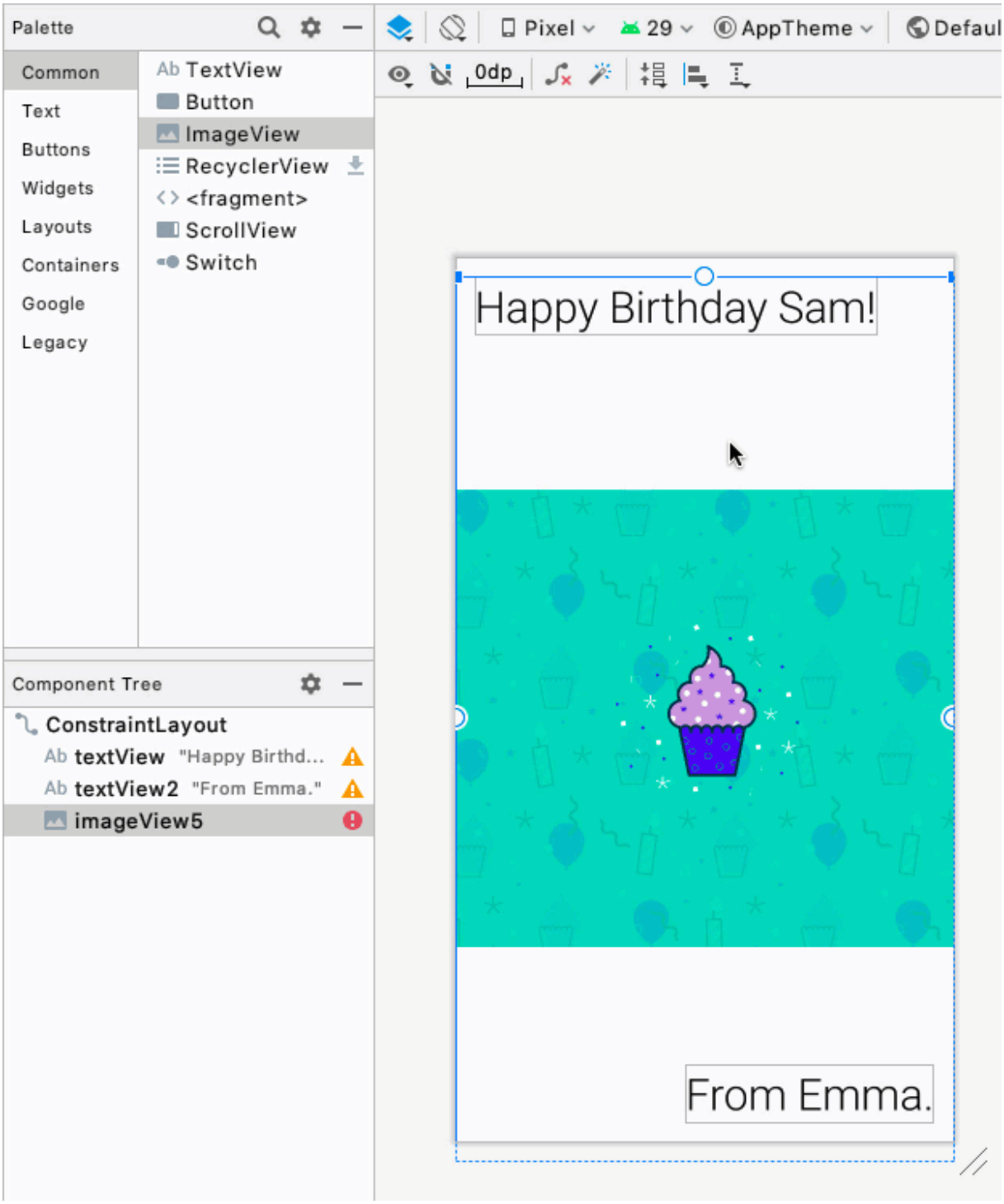
1. Kliknij i przeciągnij ImageViewwokół w **Edytorze układu** i zauważ, że podczas przeciągania wokół ekranu aplikacji w widoku **Projekt** pojawia się różowy prostokąt . Różowy prostokąt wskazuje granice ekranu do pozycjonowania ImageView.
2. Upuść ImageViewtak, aby lewa i prawa krawędź były wyrównane z różowym prostokątem. Android Studio „przyciągnie” obraz do krawędzi, gdy będziesz blisko. (Za chwilę zajmiesz się górą i dołem.)



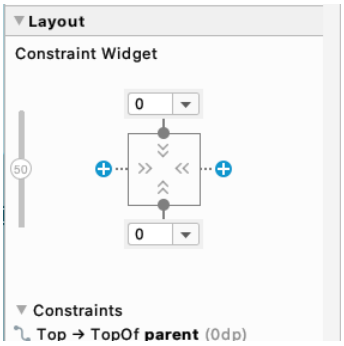
Viewsw ConstraintLayoutpotrzebie posiadania ograniczeń poziomych i pionowych, aby powiedzieć, ConstraintLayoutjak je ustawić. Dodasz je w następnej kolejności.

Uwaga: Ograniczenie w kontekście **Edytora układu** reprezentuje połączenie lub wyrównanie widoku z innym widokiem, układem macierzystym lub niewidoczną prowadnicą. Na przykład widok może być ograniczony w pewnej odległości od krawędzi kontenera, zawsze na prawo od innego widoku lub zawsze w widoku z góry wewnątrz kontenera.

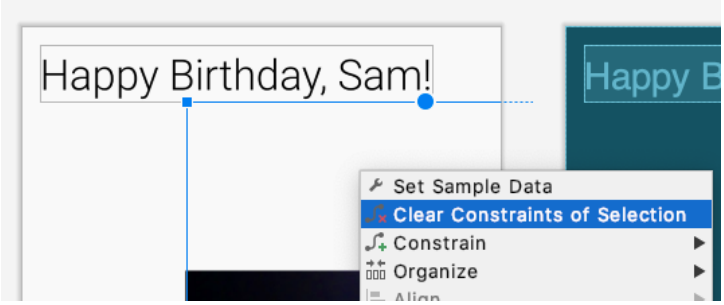
1. Przytrzymaj wskaźnik nad okręgiem u góry konturu `ImageView`, a zostanie on podświetlony innym okręgiem.
2. Przeciągnij okrąg w kierunku górnej części ekranu aplikacji, a strzałka połączy okrąg ze wskaźnikiem podczas przeciągania. Przeciągnij, aż zatrzaśnie się na górze ekranu. Dodałeś wiązanie od góry `ImageView`do góry `ConstraintLayout`.



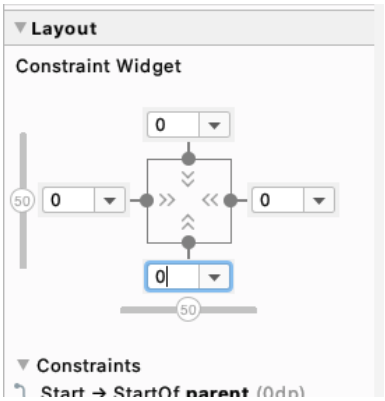
1. Dodaj wiązanie od dołu ImageViewdo dołu ConstraintLayout. Może być zbyt blisko krawędzi, aby go przeciągnąć, tak jak zrobiłeś to na górze. W takim przypadku możesz kliknąć dolny **+** w **Widżecie Ograniczeń** w oknie **Atrybuty** , aby dodać ograniczenie. Upewnij się, że margines wynosi 0.



Porada: Dodawanie ograniczeń może być trudniejsze, im więcej dodajesz Viewsna ekranie. Jeśli dodasz ograniczenie do niewłaściwego View, możesz cofnąć ten krok za pomocą Control+Z(Command+Zna Macu.) Lub kliknij prawym przyciskiem myszy, Viewco spowoduje wyświetlenie menu i wybierz **Wyczyść ograniczenia wyboru** , co spowoduje usunięcie wszystkich ograniczeń z tego View.



1. W panelu **Atrybuty** użyj **widżetu Ograniczenie** , aby dodać margines 0 z lewej i prawej strony. To automatycznie tworzy wiązanie w tym kierunku.



Obraz jest teraz wyśrodkowany, ale nie zajmuje jeszcze całego ekranu. Naprawisz to w kolejnych krokach:

1. Poniżej **widżetu Ograniczenie** w sekcji **Ograniczenia** ustaw szerokość_układu na **0dp (ograniczenie dopasowania)** . 0dp to skrót, który mówi Android Studio,

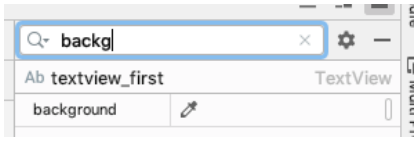
aby używał *ograniczenia dopasowania* dla szerokości ImageView. Z powodu właśnie dodanych ograniczeń powoduje to, że jest on tak szeroki jak ConstraintLayout, bez marginesów.

1. Ustaw **layout_height** na **0dp (ograniczenie dopasowania)** . Ze względu na dodane ograniczenia powoduje to, że ImageViewwysokość jest równa ConstraintLayout, minus wszelkie marginesy.

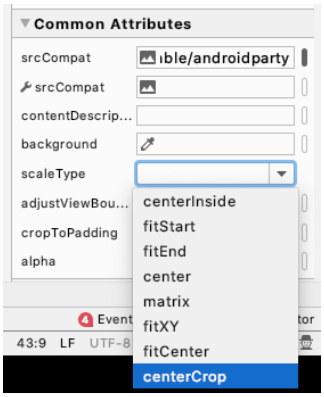
1. Jest ImageViewtak szeroki i wysoki jak ekran aplikacji, ale obraz jest wyśrodkowany w środku, ImageViewa nad i pod obrazem znajduje się kilka białych znaków. Ponieważ nie wygląda to zbyt atrakcyjnie, dostosujesz **scaleType** , ImageViewktóry mówi, jak dopasować rozmiar i wyrównać obraz. Przeczytaj więcej ScaleTypew przewodniku dla programistów . Twoja aplikacja powinna teraz wyglądać tak, jak pokazano poniżej.

1. Znajdź atrybut **scaleType** . Może być konieczne przewinięcie w dół lub wyszukanie tego atrybutu. Spróbuj ustawić różne wartości dla **scaleType** , aby zobaczyć, co robią.

Wskazówka: Aby znaleźć właściwość na liście wszystkich właściwości, kliknij ikonę lupy po prawej stronie Atrybutów i zacznij wpisywać nazwę właściwości. Android Studio pokaże tylko właściwości zawierające ten ciąg.



1. Gdy skończysz, ustaw **scaleType** na **centerCrop** , ponieważ dzięki temu obraz wypełnia ekran bez jego zniekształcania.



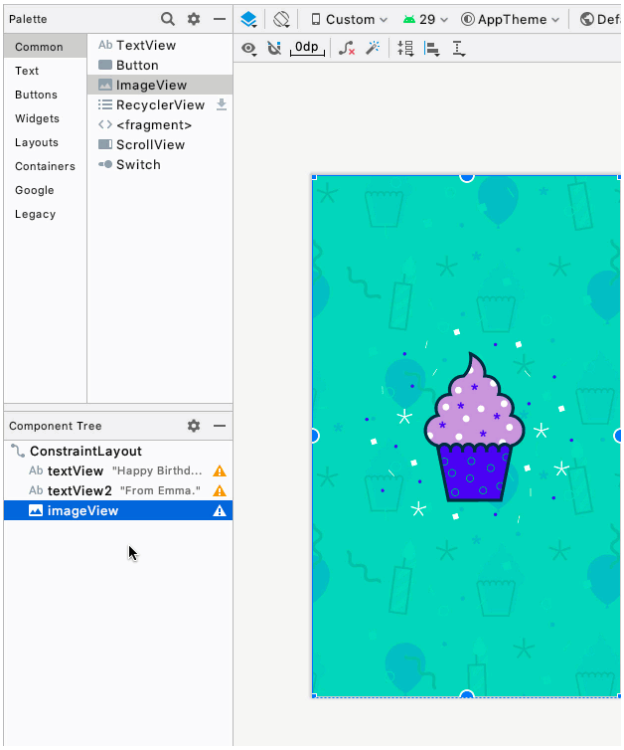
Obraz ciasta powinien teraz wypełnić cały ekran, jak pokazano poniżej.

Ale teraz nie widzisz swojego życzenia urodzinowego ani podpisu. Naprawisz to w następnym kroku.

Przesuń ImageView za tekst

Po `ImageView`wypełnieniu ekranu nie widać już tekstu. Dzieje się tak, ponieważ obraz zakrywa teraz tekst. Okazuje się, że kolejność elementów UI ma znaczenie. Dodałeś `TextViews`pierwszy, a następnie dodałeś `ImageView`, co oznacza, że znalazł się na górze. Gdy dodajesz widoki do układu, są one dodawane na końcu listy widoków i są rysowane w kolejności, w jakiej znajdują się na liście. Został `ImageView` dodany na końcu listy `Views`w `ConstraintLayout`, co oznacza, że jest rysowany jako ostatni i rysuje się nad `TextViews`. Aby to naprawić, zmienisz kolejność widoków i przeniesiesz `ImageView`na początek listy, aby była narysowana jako pierwsza.

W **drzewie komponentów** kliknij ImageViewi przeciągnij powyżej TextViewsdo tuż poniżej ConstraintLayout. Pojawia się niebieska linia z trójkątem wskazująca, dokąd ImageViewpójdzie. Upuść ImageViewtuż poniżej ConstraintLayout.



Powinien być teraz pierwszy na ImageViewliście poniżej ConstraintLayout, z następującą TextViewspo nim. Zarówno "Happy Birthday, Sam!" i „Od Emmy”. tekst powinien być teraz widoczny. (Na razie zignoruj ostrzeżenie dotyczące brakującego opisu treści).

Gratulacje! Dodałeś obraz do swojej aplikacji na Androida!

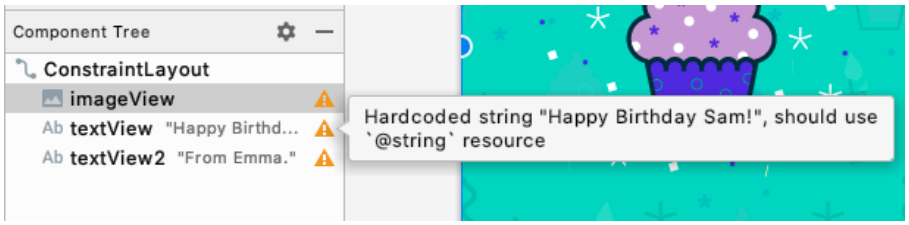
4. Zastosuj dobre praktyki kodowania

Po dodaniu `TextViews`w poprzednim ćwiczeniu z programowania Android Studio oznaczyło je trójkątami ostrzegawczymi. W tym kroku naprawisz te ostrzeżenia, a także poprawisz ostrzeżenie w `ImageView`.

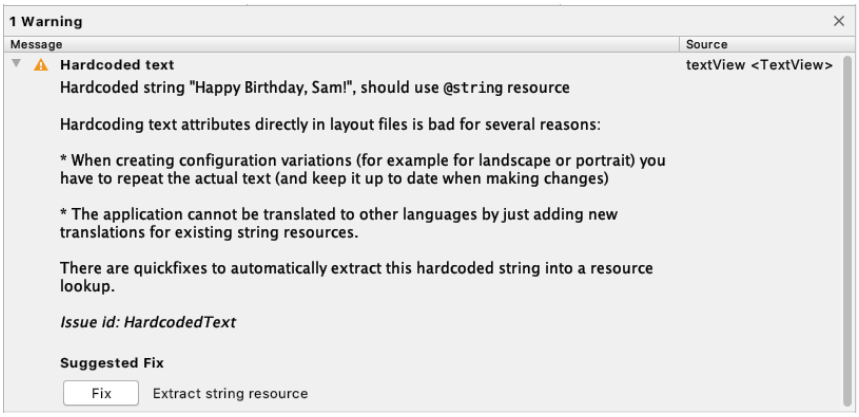
Tłumaczenie

Podczas pisania aplikacji należy pamiętać, że w pewnym momencie mogą one zostać przetłumaczone na inny język. Jak nauczyłeś się we wcześniejszym laboratorium programowania, ciąg znaków to sekwencja znaków, np. „Happy Birthday, Sam!”.

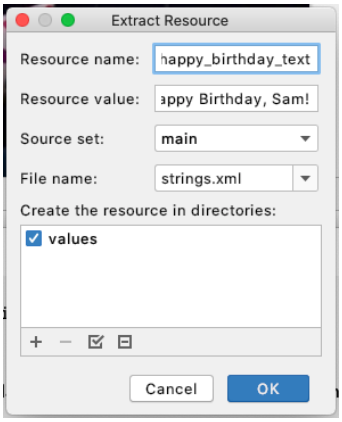
Ciąg *zakodowany* na sztywno to taki, który jest zapisywany bezpośrednio w kodzie aplikacji. Zakodowane ciągi znaków utrudniają tłumaczenie aplikacji na inne języki i utrudniają ponowne użycie ciągu w różnych miejscach aplikacji. Możesz poradzić sobie z tymi problemami, „wyodrębniając ciągi do pliku zasobów”. Oznacza to, że zamiast zakodować ciąg w kodzie, umieszczasz ciąg w pliku, nadajesz mu nazwę, a następnie używasz nazwy, gdy chcesz użyć tego ciągu. Nazwa pozostanie taka sama, nawet jeśli zmienisz ciąg lub przetłumaczysz go na inny język.



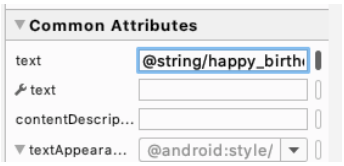
1. Kliknij pomarańczowy trójkąt obok pierwszego `TextView`z napisem „Happy Birthday, Sam!” Android Studio otwiera okno z dodatkowymi informacjami i sugerowaną poprawką. Może być konieczne przewinięcie w dół, aby zobaczyć **sugerowaną poprawkę** .



1. Kliknij przycisk **Napraw** . Android Studio otworzy okno dialogowe **Wyodrębnij zasób** . W tym oknie dialogowym możesz dostosować nazwę zasobu tekstowego i niektóre szczegóły dotyczące jego przechowywania. Nazwa **zasobu** będzie nazwą tego ciągu. Wartość **Resource** będzie samym ciągiem.
2. W oknie dialogowym **Wyodrębnij zasób** zmień **nazwę zasobu** na **happy_birthday_text** . Zasoby tekstowe powinny mieć nazwy małymi literami, a wiele słów powinno być oddzielonych podkreśleniem. Pozostałe ustawienia pozostaw wartościami domyślnymi.



1. Kliknij przycisk **OK** .
2. W panelu **Atrybuty** znajdź atrybut **tekstowy** i zauważ, że Android Studio automatycznie ustawiło go na **@string/happy_birthday_text** .



1. Otwórz **strings.xml** (**app > res > values > strings.xml**) i zauważ , że Android Studio utworzyło zasób stringów o nazwie **happy_birthday_text** .

```
<resources>
  <string name="app_name">Happy Birthday</string>
  <string name="happy_birthday_text">Happy Birthday, Sam!</string>
</resources>
```

Plik `strings.xml`zawiera listę ciągów znaków, które użytkownik zobaczy w Twojej aplikacji. Pamiętaj, że nazwa Twojej aplikacji jest również zasobem tekstowym. Umieszczając wszystkie ciągi w jednym miejscu, możesz łatwiej przetłumaczyć cały tekst w swojej aplikacji i łatwiej ponownie użyć ciągu w różnych częściach aplikacji.

1. Wykonaj te same kroki, aby wyodrębnić tekst podpisu `TextView`i nazwij zasób ciągu **podpis_tekst** .

Twój gotowy plik powinien wyglądać tak.

```
<resources>
  <string name="app_name">Happy Birthday</string>
  <string name="happy_birthday_text">Happy Birthday, Sam!</string>
  <string name="signature_text">From Emma.</string>
</resources>
```

Sprawdź swoją aplikację pod kątem ułatwień dostępu

Przestrzeganie dobrych praktyk kodowania dotyczących ułatwień dostępu umożliwia wszystkim użytkownikom, w tym użytkownikom niepełnosprawnym, łatwiejszą nawigację i interakcję z aplikacją.

Uwaga: Android zapewnia użytkownikom wiele narzędzi. Na przykład TalkBack to czytnik ekranu Google dostępny na urządzeniach z Androidem. TalkBack przekazuje użytkownikom informacje głosowe, dzięki czemu użytkownicy mogą korzystać z urządzenia bez patrzenia na ekran. Przeczytaj więcej o ułatwieniach dostępu w przewodniku dla programistów .

Android Studio udostępnia wskazówki i ostrzeżenia, które pomogą Ci uczynić Twoją aplikację bardziej dostępną.

1. W **drzewie komponentów** zwróć uwagę na pomarańczowy trójkąt obok ImageViewtego dodanego wcześniej. Jeśli najedziesz na niego kursorem, zobaczysz etykietkę z ostrzeżeniem o braku atrybutu „contentDescription" na obrazie. Opis treści może zwiększyć użyteczność Twojej aplikacji z TalkBack przez zdefiniowanie przeznaczenia elementu interfejsu użytkownika.

Jednak obraz w tej aplikacji jest dołączony tylko do celów dekoracyjnych. Zamiast ustawiać opis treści, który jest ogłaszany użytkownikowi, możesz po prostu powiedzieć TalkBack, aby pominął ten opis ImageView, ustawiając jego atrybut ImportantForAccessibility **na no** .

1. W **drzewie komponentów** wybierz ImageView.
2. W oknie **Atrybuty** , w sekcji **Wszystkie atrybuty** , znajdź wartość **ImportantForAccessibility** i ustaw ją na **no** .

Pomarańczowy trójkąt obok ImageViewznika.

1. Uruchom aplikację ponownie, aby upewnić się, że nadal działa.

Gratulacje! Dodałeś obraz do swojej aplikacji Happy Birthday, przestrzegałeś wytycznych dotyczących ułatwień dostępu i ułatwiłeś tłumaczenie na inne języki!

5. Kod rozwiązania

Kod rozwiązania dla aplikacji Happy Birthday został przesłany do GitHub, na wypadek gdybyś chciał zobaczyć kod, który otrzymaliśmy.

GitHub to usługa, która umożliwia programistom zarządzanie kodem dla ich projektów oprogramowania. Używa Git, który jest systemem kontroli wersji, który śledzi zmiany wprowadzone dla każdej wersji kodu. Jeśli kiedykolwiek widziałeś historię wersji dokumentu Google, możesz sprawdzić, kiedy i jakie zmiany zostały wprowadzone w dokumencie w przeszłości. Podobnie możesz śledzić historię wersji kodu w projekcie. Jest to bardzo pomocne, jeśli pracujesz nad projektem indywidualnie lub w zespole.

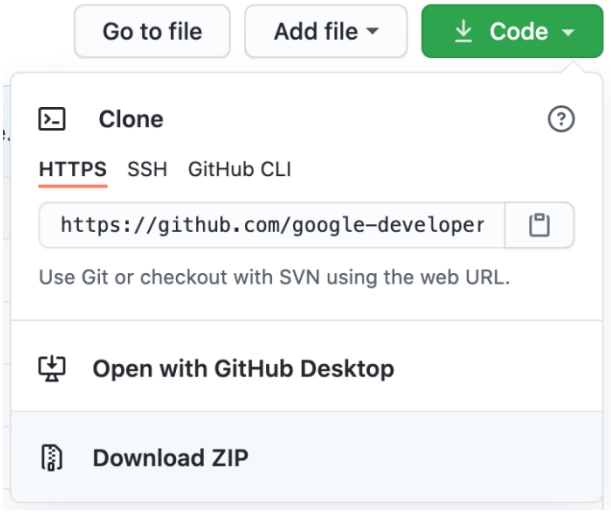
GitHub ma również stronę internetową, na której możesz przeglądać i zarządzać swoim projektem. To łącze GitHub umożliwia przeglądanie plików projektu Happy Birthday online lub pobieranie ich na komputer.

Adres URL kodu rozwiązania: https://github.com/google-developer-training/android-basics-kotlin-birthday-card-with-image-app-solution

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

Pobierz kod

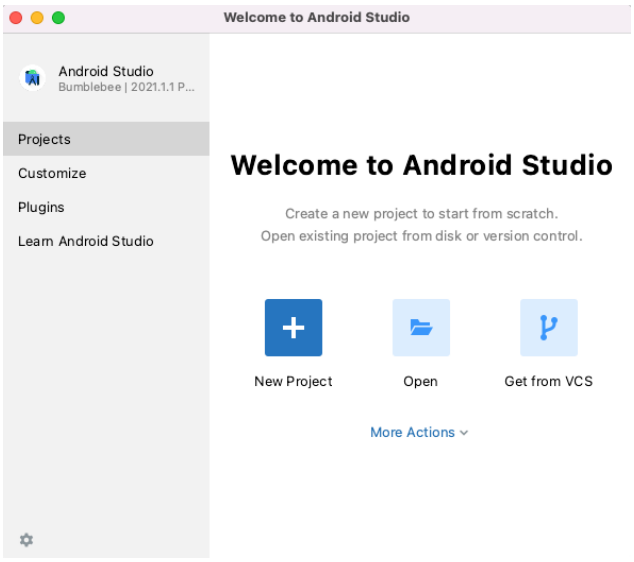
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



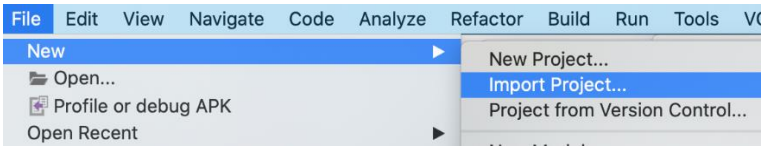
1. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
2. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
3. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego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



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

6. Podsumowanie

- Menedżer **zasobów** w Android Studio pomaga dodawać i organizować obrazy i inne zasoby.
- An `ImageView` to element interfejsu użytkownika do wyświetlania obrazów w Twojej aplikacji.
- `ImageViews` powinien zawierać opis treści, aby Twoja aplikacja była bardziej dostępna.
- Tekst wyświetlany użytkownikowi, taki jak życzenia urodzinowe, powinien zostać wyodrębniony do zasobu tekstowego, aby ułatwić przetłumaczenie aplikacji na inne języki.

7. Dowiedz się więcej

- Słownictwo dla Androida Podstawy w Kotlin
- Menedżer zasobów w Android Studio
- `ImageView`klasa
- Dostępność
- Obsługuj różne języki
- Pierwsze kroki z GitHub

8. Ćwicz na własną rękę

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

Wykonaj następujące czynności:

1. Stwórz własną aplikację kartki urodzinowej na podstawie swojego projektu.
2. Zacznij od zastanowienia się, czego `Views`będziesz potrzebować.
3. W jakiej kolejności najłatwiej byłoby je dodać?
4. Jakie obrazy musisz dodać do folderu do rysowania?

Istnieją dwa rodzaje obrazów bitmapowych powszechnie używanych w aplikacjach na Androida: pliki JPEG i pliki PNG. Pliki PNG mogą zawierać przezroczyste (puste) obszary. Przeczytaj więcej o formatach obrazów w dokumentacji dla programistów .

1. Pamiętaj, aby umieścić `Views`pierwszy z ograniczeniami i marginesami, a następnie nadać im styl.
2. Aby tekst bardziej wyróżniał się na niektórych obrazach, spróbuj poeksperymentować z parametrami **shadowColor** , **shadowDx** , **shadowDy** i **shadowRadius** .

Sprawdź swoją pracę:

Gotowa aplikacja powinna działać bez błędów i pokazywać zaprojektowaną przez Ciebie kartkę urodzinową.

Gratulacje, ukończyłeś tworzenie własnej aplikacji Kartka Urodzinowa! Podziel się swoją pracą w mediach społecznościowych i użyj hashtagu #LearningKotlin, abyśmy mogli to zobaczyć!

Ścieżka 4. Dodaj przycisk do aplikacji

Dowiedz się więcej pojęć w Kotlin — w tym klas, obiektów i warunków — aby stworzyć interaktywną aplikację dla swoich użytkowników.

Klasy i instancje obiektów w Kotlin

1. Zanim zaczniesz

Na potrzeby ćwiczeń z kodowania w tej ścieżce zbudujesz aplikację Dice Roller na Androida. Kiedy użytkownik „rzuca kostką”, zostanie wygenerowany losowy wynik. Wynik uwzględnia liczbę boków kostki. Na przykład z 6-ściennych kości można rzucić tylko wartościami od 1 do 6.

Tak będzie wyglądać finalna aplikacja.

Aby pomóc Ci skoncentrować się na nowych koncepcjach programowania dla tej aplikacji, użyjesz opartego na przeglądarce narzędzia programistycznego Kotlin do tworzenia podstawowych funkcji aplikacji. Program wyświetli Twoje wyniki na konsoli. Później zaimplementujesz interfejs użytkownika w Android Studio.

W tym pierwszym laboratorium programowania stworzysz program Kotlin, który symuluje rzucanie kostką i wyświetla losową liczbę, tak jak robiłaby to kostka.

Warunki wstępne

- Jak otwierać, edytować i uruchamiać kod na https://developer.android.com/training/kotlinplayground
- Utwórz i uruchom program Kotlin, który używa zmiennych i funkcji oraz wyświetla wynik w konsoli.
- Formatuj liczby w tekście za pomocą szablonu ciągu z `${variable}`notacją.

Czego się nauczysz

- Jak programowo generować liczby losowe, aby symulować rzuty kostką.
- Jak ustrukturyzować kod, tworząc `Dice`klasę ze zmienną i metodą.
- Jak utworzyć instancję obiektu klasy, modyfikować jej zmienne i wywoływać jej metody.

Co zbudujesz

- Program Kotlin w opartym na przeglądarce narzędziu programistycznym Kotlin, który może wykonać losowy rzut kostką.

Czego potrzebujesz

- Komputer z połączeniem internetowym

2. Rzuć losowe liczby

Gry często mają w sobie element losowy. Możesz zdobyć losową nagrodę lub przejść o losową liczbę kroków na planszy. W codziennym życiu możesz używać losowych cyfr i liter, aby generować bezpieczniejsze hasła!

Zamiast rzucać kostką, możesz napisać program, który symuluje dla ciebie rzucanie kośćmi. Za każdym razem, gdy rzucasz kostką, wynikiem może być dowolna liczba z zakresu możliwych wartości. Na szczęście nie musisz budować własnego generatora liczb losowych dla takiego programu. Większość języków programowania, w tym Kotlin, ma wbudowany sposób generowania liczb losowych. W tym zadaniu użyjesz kodu Kotlin do wygenerowania liczby losowej.

Skonfiguruj swój kod startowy

1. W przeglądarce otwórz stronę https://developer.android.com/training/kotlinplayground .
2. Usuń cały istniejący kod w edytorze kodu i zastąp go poniższym kodem. Jest to `main()`funkcja, z którą pracowałeś we wcześniejszych ćwiczeniach z programowania.

```
fun main() {

}
```

Użyj funkcji losowej!

Aby rzucić kostką, potrzebujesz sposobu, aby przedstawić wszystkie prawidłowe wartości rzutu kośćmi. W przypadku zwykłych kości sześciościennych dopuszczalne rzuty to: 1, 2, 3, 4, 5 i 6.

Wcześniej dowiedziałeś się, że istnieją typy danych, takie jak `Int`liczby całkowite i `String`tekst. `IntRange`jest innym typem danych i reprezentuje zakres liczb całkowitych od punktu początkowego do punktu końcowego. `IntRange`jest odpowiednim typem danych do reprezentowania możliwych wartości, jakie może wytworzyć rzut kostką.

1. Wewnątrz `main()`funkcji zdefiniuj zmienną jako o `val`nazwie `diceRange`. Przypisz go do `IntRange`liczby od 1 do 6, reprezentującej zakres liczb całkowitych, którymi można rzucić kostką 6-ścienną.

```
val diceRange = 1..6
```

Możesz powiedzieć, że `1..6`jest to zakres Kotlina, ponieważ ma numer początkowy, dwie kropki, po których następuje numer końcowy (bez spacji pomiędzy). Inne przykłady zakresów liczb całkowitych dotyczą `2..5`liczb od 2 do 5 oraz `100..200`liczb od 100 do 200.

Wskazówka: Zwróć uwagę, że nie mówi `IntRange`to w tej definicji, w ten sam sposób, w jaki nie trzeba było określać `Int`ani `String`podczas tworzenia zmiennej dla liczby całkowitej lub łańcucha. W większości przypadków system może określić, jaki typ danych chcesz.

Na przykład system interpretuje to:

```
val diceRange = 1..6
```

jak to:

```
val diceRange: IntRange = 1..6
```

Podobnie do tego, jak wywołanie `println()`nakazuje systemowi wydrukowanie podanego tekstu, możesz użyć funkcji wywołanej `random()`do wygenerowania i zwrócenia losowej liczby dla danego zakresu. Tak jak poprzednio, możesz przechowywać wynik w zmiennej.

1. Wewnątrz `main()`zdefiniuj zmienną jako o `val`nazwie `randomNumber`.
2. Make `randomNumber`mają wartość wyniku sprawdzania `random()`w `diceRange`zakresie, jak pokazano poniżej.

```
val randomNumber = diceRange.random()
```

Zauważ, że wołasz używając kropki lub kropki pomiędzy zmienną a wywołaniem funkcji `random()`. `diceRange`Możesz to przeczytać jako "generowanie losowej liczby z `diceRange`". Wynik jest następnie przechowywany w `randomNumber`zmiennej.

1. Aby zobaczyć losowo wygenerowaną liczbę, użyj notacji formatowania ciągu (zwanej również „szablonem ciągu”) `${randomNumber}`, aby ją wydrukować, jak pokazano poniżej.

```
println("Random number: ${randomNumber}")
```

Twój gotowy kod powinien wyglądać tak.

```
fun main() {
   val diceRange = 1..6
   val randomNumber = diceRange.random()
   println("Random number: ${randomNumber}")
```

```
}
```

1. Uruchom swój kod kilka razy. Za każdym razem powinieneś zobaczyć dane wyjściowe jak poniżej, z różnymi liczbami losowymi.

```
Random number: 4
```

Wskazówki dotyczące zakresów :

- Zakresy mogą zawierać się między dowolnymi liczbami całkowitymi. Prawidłowe zakresy to: `3..46`, `0..270`, `-6..+6`, `-10..-4`.
- Możesz wywoływać funkcje bezpośrednio w zakresie, na przykład: `(1..6).random()`.

3. Stwórz klasę kości

Kiedy rzucasz kośćmi, w twoich rękach są one prawdziwymi przedmiotami. Chociaż kod, który właśnie napisałeś, działa doskonale, trudno sobie wyobrazić, że dotyczy rzeczywistych kości. Zorganizowanie programu tak, aby był bardziej podobny do rzeczy, które reprezentuje, ułatwia zrozumienie. Tak więc fajnie byłoby mieć zautomatyzowane kości, którymi można rzucać!

Wszystkie kości działają zasadniczo tak samo. Mają te same właściwości, na przykład boki, i takie samo zachowanie, na przykład możliwość toczenia. W Kotlinie możesz stworzyć programowy plan kości, który mówi, że kości mają boki i mogą rzucać losową liczbą. Ten plan nazywa się *klasą* .

Z tej klasy możesz następnie tworzyć rzeczywiste obiekty kości, zwane *instancjami obiektów* . Na przykład możesz utworzyć kostkę 12-ścienną lub 4-ścienną.

Wskazówka: Klasa jest podobna do tego , w jaki sposób plany architekta nie są domem; są instrukcjami, jak zbudować dom. Dom jest rzeczywistą rzeczą lub *instancją obiektu* stworzoną zgodnie z planem.

Uwaga: Organizowanie wszystkiego o kostkach w klasę nazywa się enkapsulacją. *Enkapsulacja* to duże, wymyślne słowo, ale oznacza to tylko to, że możesz zamknąć funkcjonalność, która jest logicznie powiązana w jednym miejscu.

Zdefiniuj klasę kości

W kolejnych krokach zdefiniujesz nową klasę, `Dice`która będzie reprezentować rzucaną kostką.

1. Aby zacząć od nowa, wyczyść kod w `main()`funkcji, aby otrzymać kod, jak pokazano poniżej.

```
fun main() {

}
```

1. Poniżej tej `main()`funkcji dodaj pustą linię, a następnie dodaj kod, aby utworzyć `Dice`klasę. Jak pokazano poniżej, zacznij od słowa kluczowego `class`, po którym następuje nazwa klasy, a następnie otwierający i zamykający nawias klamrowy. Zostaw spację między nawiasami klamrowymi, aby umieścić kod dla klasy.

```
class Dice {
```

```
}
```

Notatka:

- Podobnie jak przy użyciu `fun`słowa kluczowego w Kotlinie do tworzenia nowej funkcji, użyj `class`słowa kluczowego, aby utworzyć nową klasę.
- Możesz wybrać dowolną nazwę dla `class`, ale jest to pomocne, jeśli nazwa wskazuje, co reprezentuje klasa. Zgodnie z konwencją nazwa klasy jest zapisywana w formacie Upper Camel Case (zwanym również przypadkiem Pascal Caseing). Na przykład: `Car`, `ParkingMeter`i `CustomerRecord`to poprawne nazwy klas i możesz zgadywać, co reprezentują.

Wewnątrz definicji klasy możesz określić jedną lub więcej właściwości klasy za pomocą zmiennych. Prawdziwe kości mogą mieć kilka boków, kolor lub wagę. W tym zadaniu skupisz się na właściwości liczby boków kostki.

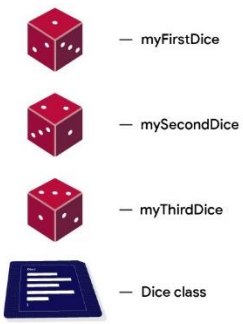
1. Wewnątrz `Dice` klasy dodaj `var`wywołanie `sides`określające liczbę stron, które będą miały twoje kości. Ustaw `sides`na 6.

```
class Dice {
  var sides = 6
}
```

Otóż to. Masz teraz bardzo prostą klasę reprezentującą kości.

Utwórz instancję klasy Dice

Dzięki tej `Dice`klasie masz plan tego, czym jest kostka. Aby mieć rzeczywistą kostkę w swoim programie, musisz utworzyć `Dice`instancję obiektu. (A gdybyś potrzebował trzech kości, utworzyłbyś trzy instancje obiektów).



1. Aby utworzyć instancję obiektu `Dice`, w `main()`funkcji utwórz `val`wywołaną `myFirstDice`i zainicjuj ją jako instancję `Dice`klasy. Zwróć uwagę na nawiasy po nazwie klasy, które oznaczają, że tworzysz nową instancję obiektu z klasy.

```
fun main() {
  val myFirstDice = Dice()
```

```
}
```

Teraz, gdy masz `myFirstDice`obiekt, rzecz zrobioną z planu, możesz uzyskać dostęp do jego właściwości. Jedyną właściwością `Dice`jest jego `sides`. Dostęp do właściwości uzyskuje się za pomocą „notacji kropkowej”. Tak więc, aby uzyskać dostęp do `sides`właściwości `myFirstDice`, wywołujesz, `myFirstDice.sides`co jest wymawiane jako „ `myFirstDice`kropka `sides`”.

1. Poniżej deklaracji `myFirstDice`, dodaj instrukcję, aby wypisać `println()`liczbę`sidesmyFirstDice.`

```
println(myFirstDice.sides)
```

Uwaga: wcześniej użyłeś notacji z kropką, dzwoniąc do diceRange.random(). Uogólniając, możesz myśleć o notacji kropkowej jako mówiącej „na czymś-kropka-wykonaj jakąś akcję”. Na przykład tutaj, z myFirstDice.sides, „w tym przypadku pobierz sideswłaściwość”.

Twój kod powinien wyglądać tak.

```
fun main() {
  val myFirstDice = Dice()
  println(myFirstDice.sides)
}

class Dice {
  var sides = 6
}
```

1. Uruchom swój program i powinien wypisać liczbę `sides`zdefiniowaną w `Dice`klasie.

```
6
```

Masz teraz `Dice`klasę i rzeczywistą kostkę `myFirstDice`z 6 `sides`.

Rzućmy kostką!

Rzuć kostką

Wcześniej używałeś funkcji do wykonywania czynności drukowania warstw ciasta. Rzucanie kostką to również czynność, którą można zaimplementować jako funkcję. A ponieważ można rzucać wszystkimi kośćmi, możesz dodać do nich funkcję wewnątrz `Dice`klasy. Funkcja zdefiniowana wewnątrz klasy jest również nazywana *metodą* .

1. W `Dice`klasie pod `sides`zmienną wstawiamy pustą linię, a następnie tworzymy nową funkcję do rzucania kostką. Zacznij od słowa kluczowego Kotlin `fun`, po którym następuje nazwa metody , następnie nawiasy `()`, a następnie otwierające i zamykające nawiasy klamrowe `{}`. Możesz zostawić pustą linię między nawiasami klamrowymi, aby zrobić miejsce na więcej kodu, jak pokazano poniżej. Twoja klasa powinna wyglądać tak.

```
class Dice {
  var sides = 6

  fun roll() {
```

```
  }
}
```

Uwaga: Możesz nazwać tę metodę w dowolny sposób, ale warto nadać jej nazwę wskazującą, jakie działanie wykonuje. Konwencja nazewnictwa funkcji i metod polega na rozpoczynaniu się od małej litery, używaniu wielbłąda i rozpoczynaniu od czasownika czynności, jeśli to możliwe.

Kiedy rzucasz kostką sześciościenną, otrzymujesz losową liczbę z przedziału od 1 do 6.

1. Wewnątrz `roll()`metody utwórz `val randomNumber`. Przypisz mu losową liczbę z `1..6`zakresu. Użyj notacji z kropką, aby sprawdzić `random()`zakres.

```
val randomNumber = (1..6).random()
```

1. Po wygenerowaniu losowej liczby wydrukuj ją na konsoli. Twoja gotowa `roll()`metoda powinna wyglądać jak poniższy kod.

```
fun roll() {
    val randomNumber = (1..6).random()
    println(randomNumber)
}
```

1. Aby faktycznie wprowadzić `myFirstDice`, w `main()`programie , wywołaj `roll()`metodę on `myFirstDice`. Wywołujesz metodę za pomocą „notacji kropkowej”. Tak więc, aby wywołać `roll()`metodę `myFirstDice`, wpisz, `myFirstDice.roll()`co jest wymawiane jako " `myFirstDice`kropka `roll()`".

```
myFirstDice.roll()
```

Twój ukończony kod powinien wyglądać tak.

```
fun main() {
    val myFirstDice = Dice()
    println(myFirstDice.sides)
    myFirstDice.roll()
}

class Dice {
    var sides = 6

    fun roll() {
        val randomNumber = (1..6).random()
        println(randomNumber)
    }
}
```

1. Uruchom swój kod! Powinieneś zobaczyć wynik losowego rzutu kostką poniżej liczby boków. Uruchom swój kod kilka razy i zauważ, że liczba boków pozostaje taka sama, a wartość rzutu kostką zmienia się.

```
6
4
```

Gratulacje! Zdefiniowałeś `Dice`klasę ze `sides`zmienną i `roll()`funkcją. W `main()`funkcji utworzyłeś nową `Dice`instancję obiektu, a następnie wywołałeś `roll()`na niej metodę w celu wygenerowania liczby losowej.

4. Zwróć wartość swojego rzutu kostką

Obecnie wypisujesz wartość funkcji `randomNumber`w swojej `roll()`funkcji i to działa świetnie! Ale czasami bardziej przydatne jest zwrócenie wyniku funkcji do tego, co nazywa się funkcją. Na przykład możesz przypisać wynik `roll()`metody do zmiennej, a następnie przenieść gracza o tę kwotę! Zobaczmy, jak to się robi.

1. W `main()`zmodyfikuj linię, która mówi `myFirstDice.roll()`. Utwórz o `val`nazwie `diceRoll`. Ustaw ją równą wartości zwracanej przez `roll()`metodę.

```
val diceRoll = myFirstDice.roll()
```

To jeszcze nic nie robi, ponieważ jeszcze `roll()`niczego nie zwraca. Aby ten kod działał zgodnie z przeznaczeniem, `roll()`musi coś zwrócić.

W poprzednich ćwiczeniach z programowania nauczyłeś się, że musisz określić typ danych dla argumentów wejściowych do funkcji. W ten sam sposób musisz określić typ danych dla danych zwracanych przez funkcję.

1. Zmień `roll()`funkcję, aby określić, jaki typ danych zostanie zwrócony. W tym przypadku liczbą losową jest `Int`, więc typem zwracanym jest `Int`. Składnia określania zwracanego typu to: Po nazwie funkcji, po nawiasach, dodaj dwukropek, spację, a następnie `Int`słowo kluczowe dla zwracanego typu funkcji. Definicja funkcji powinna wyglądać jak w poniższym kodzie.

```
fun roll(): Int {
```

1. Uruchom ten kod. Zobaczysz błąd w widoku **problemów** . To mówi:

```
A ‘return' expression required in a function with a block body (‘{...}')
```

Zmieniłeś definicję funkcji tak, aby zwracała `Int`, ale system narzeka, że Twój kod w rzeczywistości nie zwraca `Int`. „Treść bloku” lub „treść funkcji” odnosi się do kodu między nawiasami klamrowymi funkcji. Możesz naprawić ten błąd, zwracając wartość z funkcji za pomocą `return`instrukcji na końcu treści funkcji.

1. W `roll()`programie usuń `println()`instrukcję i zastąp ją `return`instrukcją for `randomNumber`. Twoja `roll()`funkcja powinna wyglądać jak poniższy kod.

```
fun roll(): Int {
    val randomNumber = (1..6).random()
    return randomNumber
```

```
}
```

1. W `main()`usuń nadruk na bokach kości.
2. Dodaj oświadczenie, aby wydrukować wartość `sides`i `diceRoll`w zdaniu informacyjnym. Twoja ukończona `main()`funkcja powinna wyglądać podobnie do poniższego kodu.

```
fun main() {
   val myFirstDice = Dice()
   val diceRoll = myFirstDice.roll()
   println("Your ${myFirstDice.sides} sided dice rolled ${diceRoll}!")
}
```

1. Uruchom swój kod, a wynik powinien wyglądać tak.

```
Your 6 sided dice rolled 4!
```

Oto cały Twój dotychczasowy kod.

```
fun main() {
   val myFirstDice = Dice()
   val diceRoll = myFirstDice.roll()
   println("Your ${myFirstDice.sides} sided dice rolled ${diceRoll}!")
}


class Dice {
   var sides = 6

   fun roll(): Int {
      val randomNumber = (1..6).random()
      return randomNumber
   }
}
```

5. Zmień liczbę boków na swoich kostkach

Nie wszystkie kości mają 6 stron! Kości mają różne kształty i rozmiary: 4 boki, 8 boków, do 120 boków!

1. W swojej `Dice`klasie, w swojej `roll()`metodzie, zmień zakodowane na sztywno, `1..6`aby użyć `sides`zamiast tego, aby zakres, a tym samym losowa liczba, zawsze była odpowiednia dla liczby boków.

```
val randomNumber = (1..sides).random()
```

1. W `main()`funkcji poniżej i po wydrukowaniu kostek należy zmienić `sides`na `myFirstDice`20.

```
myFirstDice.sides = 20
```

1. Skopiuj i wklej istniejące oświadczenie print poniżej po zmianie liczby stron.
2. Zastąp drukowanie of `diceRoll`drukowaniem wyniku wywołania `roll()`metody on `myFirstDice`.

```
println("Your ${myFirstDice.sides} sided dice rolled ${myFirstDice.roll()}!")
```

Twój program powinien wyglądać tak.

```
fun main() {

  val myFirstDice = Dice()
  val diceRoll = myFirstDice.roll()
  println("Your ${myFirstDice.sides} sided dice rolled ${diceRoll}!")

  myFirstDice.sides = 20
  println("Your ${myFirstDice.sides} sided dice rolled ${myFirstDice.roll()}!")
}

class Dice {
  var sides = 6

  fun roll(): Int {
    val randomNumber = (1..sides).random()
    return randomNumber
  }
}
```

1. Uruchom swój program, a powinieneś zobaczyć komunikat dla kości 6-ściennych i drugi komunikat dla kości 20-ściennych.

```
Your 6 sided dice rolled 3!
Your 20 sided dice rolled 15!
```

6. Dostosuj swoje kości

Ideą zajęć jest reprezentowanie rzeczy, często czegoś fizycznego w prawdziwym świecie. W tym przypadku `Dice`klasa reprezentuje fizyczną kość. W prawdziwym świecie kości nie mogą zmieniać liczby stron. Jeśli chcesz mieć inną liczbę boków, musisz zdobyć inną kostkę. Programowo oznacza to, że zamiast zmieniać właściwość sides istniejącej `Dice`instancji obiektu, należy utworzyć nową instancję obiektu kości z potrzebną liczbą stron.

W tym zadaniu zmodyfikujesz `Dice`klasę, aby móc określić liczbę boków podczas tworzenia nowej instancji. Zmień `Dice`definicję klasy, aby móc podać liczbę boków. Jest to podobne do tego, jak funkcja może akceptować argumenty wejściowe.

1. Zmodyfikuj `Dice`definicję klasy, aby zaakceptować liczbę całkowitą o nazwie `numSides`. Kod wewnątrz Twojej klasy nie ulega zmianie.

```
class Dice(val numSides: Int) {
  // Code inside does not change.
}
```

1. Wewnątrz `Dice`klasy usuń `sides`zmienną, ponieważ możesz teraz używać `numSides`.
2. Popraw także używany zakres `numSides`.

Twoja `Dice`klasa powinna wyglądać tak.

```
class Dice (val numSides: Int) {

  fun roll(): Int {
    val randomNumber = (1..numSides).random()
    return randomNumber
  }
}
```

Jeśli uruchomisz ten kod, zobaczysz wiele błędów, ponieważ musisz zaktualizować `main()`, aby pracować ze zmianami w `Dice`klasie.

1. W `main()`, aby utworzyć `myFirstDice`z 6 stronami, musisz teraz podać liczbę stron jako argument do `Dice`klasy, jak pokazano poniżej.

```
  val myFirstDice = Dice(6)
```

1. W instrukcji print zmień `sides`na `numSides`.
2. Poniżej usuń kod, który zmienia `sides`się na 20, ponieważ ta zmienna już nie istnieje.
3. Usuń również `println`oświadczenie znajdujące się pod nim.

Twoja `main()`funkcja powinna wyglądać jak poniższy kod, a jeśli ją uruchomisz, nie powinno być żadnych błędów.

```
fun main() {
  val myFirstDice = Dice(6)
  val diceRoll = myFirstDice.roll()
  println("Your ${myFirstDice.numSides} sided dice rolled ${diceRoll}!")
}
```

1. Po wydrukowaniu pierwszego rzutu kostką, dodaj kod, aby utworzyć i wydrukować drugi `Dice`obiekt o nazwie `mySecondDice`20 stron.

```
val mySecondDice = Dice(20)
```

1. Dodaj instrukcję print, która wyświetla i wyświetla zwróconą wartość.

```
println("Your ${mySecondDice.numSides} sided dice rolled  ${mySecondDice.roll()}!")
```

1. Twoja ukończona `main()`funkcja powinna wyglądać tak.

```
fun main() {
   val myFirstDice = Dice(6)
   val diceRoll = myFirstDice.roll()
   println("Your ${myFirstDice.numSides} sided dice rolled ${diceRoll}!")

   val mySecondDice = Dice(20)
   println("Your ${mySecondDice.numSides} sided dice rolled ${mySecondDice.roll()}!")
}

class Dice (val numSides: Int) {

   fun roll(): Int {
      val randomNumber = (1..numSides).random()
      return randomNumber
   }
}
```

1. Uruchom gotowy program, a wynik powinien wyglądać tak.

```
Your 6 sided dice rolled 5!
Your 20 sided dice rolled 7!
```

7. Zastosuj dobre praktyki kodowania

Podczas pisania kodu zwięzłość jest lepsza. Możesz pozbyć się `randomNumber`zmiennej i bezpośrednio zwrócić losową liczbę.

1. Zmień `return`instrukcję, aby bezpośrednio zwracała liczbę losową.

```
fun roll(): Int {
   return (1..numSides).random()
}
```

W drugiej instrukcji print umieszczasz wywołanie, aby uzyskać liczbę losową w szablonie ciągu. Możesz pozbyć się `diceRoll`zmiennej, robiąc to samo w pierwszej instrukcji print.

1. Wywołaj `myFirstDice.roll()`szablon ciągu i usuń `diceRoll`zmienną. Pierwsze dwie linie twojego `main()`kodu wyglądają teraz tak.

```
val myFirstDice = Dice(6)
println("Your ${myFirstDice.numSides} sided dice rolled ${myFirstDice.roll()}!")
```

1. Uruchom swój kod i nie powinno być żadnej różnicy w danych wyjściowych.

Uwaga: zmiana kodu, aby był krótszy, bardziej wydajny lub łatwiejszy do odczytania i zrozumienia, nazywa się *refaktoryzacją* . To jak pisanie dokumentu, w którym piszesz pierwszą wersję roboczą zawierającą wszystkie informacje, a następnie edytujesz i udoskonalasz swoje słowa.

To jest twój ostateczny kod po *refaktoryzacji* .

```
fun main() {
   val myFirstDice = Dice(6)
   println("Your ${myFirstDice.numSides} sided dice rolled ${myFirstDice.roll()}!")

   val mySecondDice = Dice(20)
   println("Your ${mySecondDice.numSides} sided dice rolled ${mySecondDice.roll()}!")
}

class Dice (val numSides: Int) {

   fun roll(): Int {
      return (1..numSides).random()
   }
}
```

8. Kod rozwiązania

```
fun main() {
   val myFirstDice = Dice(6)
   println("Your ${myFirstDice.numSides} sided dice rolled ${myFirstDice.roll()}!")

   val mySecondDice = Dice(20)
   println("Your ${mySecondDice.numSides} sided dice rolled ${mySecondDice.roll()}!")
}

class Dice (val numSides: Int) {

   fun roll(): Int {
      return (1..numSides).random()
   }
}
```

9. Podsumowanie

- Wywołaj `random()`funkcję na an, `IntRange`aby wygenerować liczbę losową:`(1..6).random()`
- Klasy są jak plan obiektu. Mogą mieć właściwości i zachowania, zaimplementowane jako zmienne i funkcje.
- Instancja klasy reprezentuje obiekt, często obiekt fizyczny, taki jak kostka do gry. Możesz wywołać akcje na obiekcie i zmienić jego atrybuty.
- Możesz podać wartości do klasy podczas tworzenia instancji. Na przykład: `class Dice(val numSides: Int)`a następnie utwórz instancję za pomocą `Dice(6)`.
- Funkcje mogą coś zwrócić. Określ typ danych do zwrócenia w definicji funkcji i użyj `return`instrukcji w treści funkcji, aby coś zwrócić. Na przykład:`fun example(): Int { return 5 }`

10. Dowiedz się więcej

- Słownictwo dla Androida Podstawy w Kotlin
- Generowanie liczb losowych (Wikipedia)
- Zaskakujące wyzwanie polegające na zaprojektowaniu 120-ściennych kości
- Zajęcia w Kotlinie
- Deklaracje funkcji w Kotlin
- Zwracanie wartości z funkcji

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:

- Daj swojej `Dice`klasie kolejny atrybut koloru i stwórz wiele instancji kości z różną liczbą boków i kolorów!
- Stwórz `Coin`klasę, daj jej możliwość odwracania się, stwórz instancję klasy i odwróć kilka monet! Jak wykorzystałbyś `random()`funkcję z zasięgiem, aby wykonać rzut monetą?

Stwórz interaktywną aplikację Dice Roller

1. Zanim zaczniesz

W tym ćwiczeniu z kodowania stworzysz aplikację Dice Roller na Androida, w której użytkownicy będą mogli kliknąć ikonę `Button`w aplikacji, aby rzucić kostką. Wynik rzutu zostanie pokazany `TextView`na ekranie.

Użyjesz **Edytora układu** w Android Studio, aby zbudować układ swojej aplikacji, a następnie napiszesz kod Kotlin, co się stanie po `Button`kliknięciu.

Warunki wstępne

- Jak stworzyć i uruchomić „Hello, World!” w Android Studio.
- Zaznajomiony z używaniem `TextViews`w aplikacji.
- Jak zmienić atrybuty `TextView`w **Edytorze układu** .
- Jak wyodrębnić tekst do zasobu ciągu, aby ułatwić tłumaczenie aplikacji i ponowne użycie ciągów.
- Podstawy programowania Kotlin

Czego się nauczysz

- Jak dodać a `Button`do aplikacji na Androida.

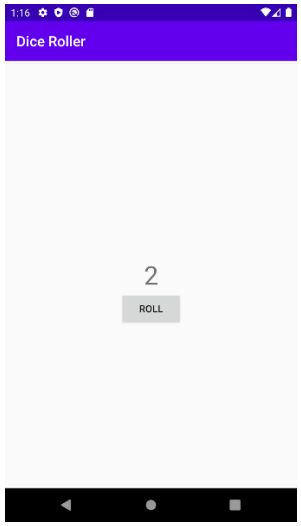
- Jak dodać zachowanie po `Button`stuknięciu w aplikacji.
- Jak otworzyć i zmodyfikować `Activity`kod aplikacji.
- Jak wyświetlić `Toast`wiadomość.
- Jak zaktualizować zawartość przez chwilę, `TextView`gdy aplikacja jest uruchomiona.

Co zbudujesz

- Aplikacja Dice Roller na Androida, która ma `Button`rzucać kostką i aktualizuje tekst na ekranie o wynik rzutu.

Czego potrzebujesz

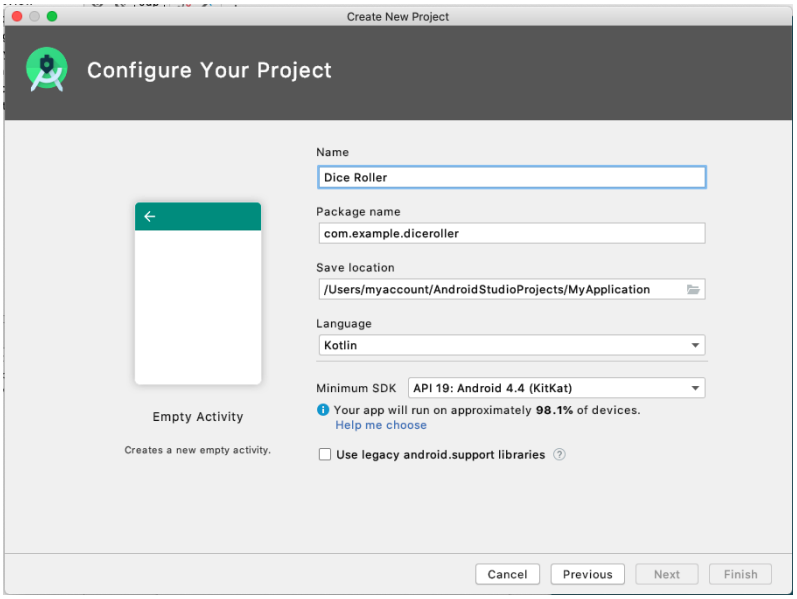
- Komputer z zainstalowanym Android Studio.
- Oto jak aplikacja będzie wyglądać po ukończeniu tego ćwiczenia z programowania.



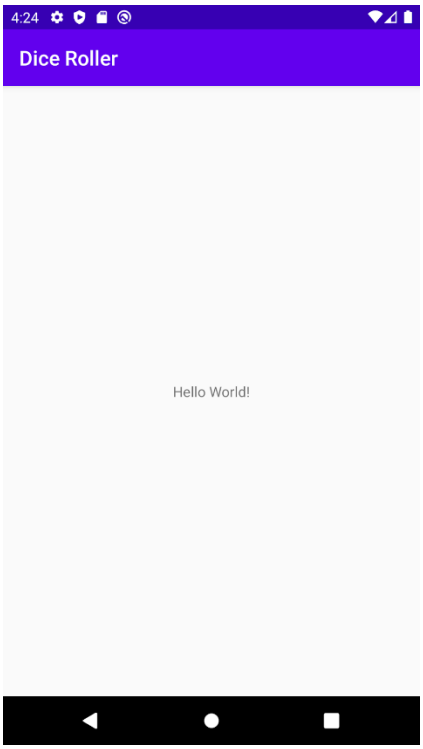
2. Skonfiguruj swoją aplikację

Utwórz projekt pustej aktywności

1. Jeśli masz już istniejący projekt otwarty w Android Studio, przejdź do **Plik > Nowy > Nowy projekt...** , aby otworzyć ekran **Utwórz nowy projekt .**
2. W **Create New Project** stwórz nowy projekt Kotlin, korzystając z szablonu **Empty Activity .**
3. Wywołaj aplikację „Dice Roller” z minimalnym poziomem API 19 (KitKat).



1. Uruchom nową aplikację i powinna wyglądać tak.

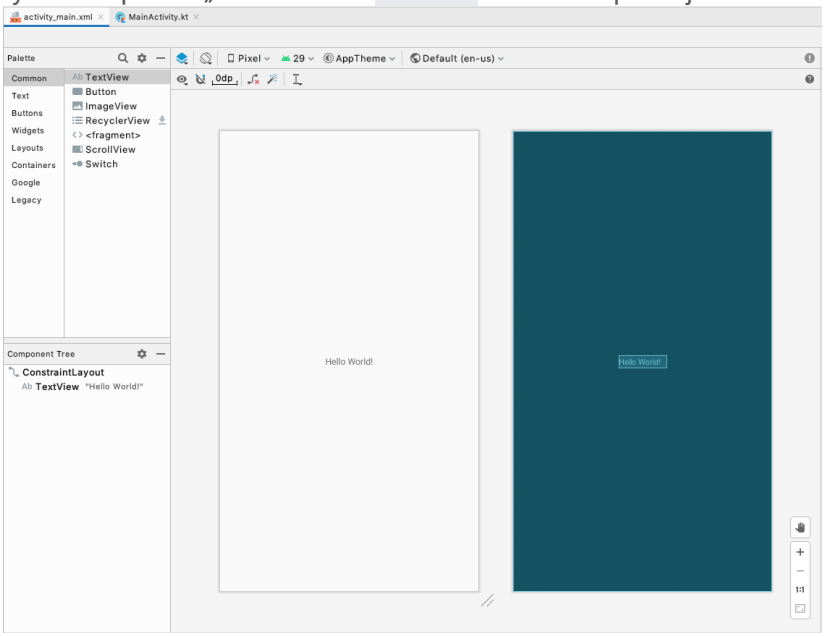


3. Utwórz układ aplikacji

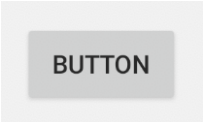
Otwórz Edytor układu

1. W oknie **projektu** kliknij dwukrotnie activity_main.xml(**app >** **res** > **layout** > **activity_main.xml**), aby go otworzyć. Powinieneś zobaczyć **Edytor układu** ,

tylko z napisem „Hello World” TextVieww środku aplikacji.

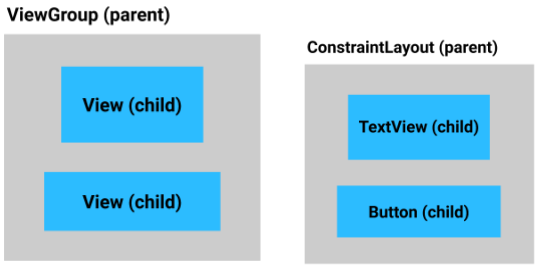


Następnie dodasz Buttondo swojej aplikacji. A Buttonto element interfejsu użytkownika (UI) w systemie Android, który użytkownik może dotknąć, aby wykonać akcję.



W tym zadaniu dodajesz Buttonponiżej „Hello World” TextView. The TextViewand the Buttonbędzie znajdować się w ConstraintLayout, który jest typem ViewGroup.

Gdy są Viewsw obrębie ViewGroup, Viewssą uważane za *dzieci rodzica*ViewGroup . W przypadku Twojej aplikacji znak TextViewi Buttonbędzie uważany za elementy podrzędne rodzica ConstraintLayout.

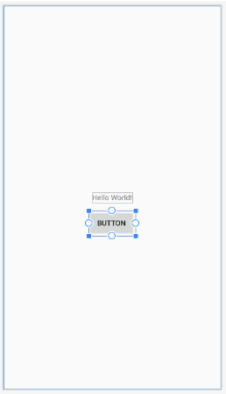


Dodaj Buttonjako dziecko istniejącego ConstraintLayoutw swojej aplikacji.

Uwaga: Podobnie jak w drzewie genealogicznym, w *hierarchii* widoków, widoki rodzicielskie mogą same być widokami potomnymi, a widoki potomne mogą być rodzicami innych dzieci.

Dodaj przycisk do układu

1. Przeciągnij a Buttonz **Palety** do widoku **Projekt** , umieszczając go poniżej „Witaj świecie” TextView.

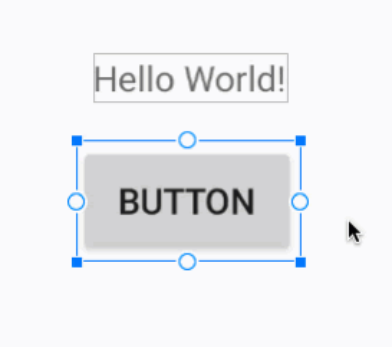


1. Sprawdź, czy poniżej **palety** w **drzewie komponentów**Button i TextViewsą wymienione pod ConstraintLayout(jako elementy podrzędne ConstraintLayout).
2. Zwróć uwagę na błąd, że Buttonnie jest ograniczony. Ponieważ Buttonznajduje się w a ConstraintLayout, musisz ustawić ograniczenia w pionie i poziomie, aby go ustawić.

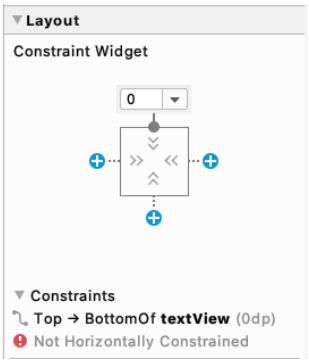
Ustaw przycisk

W tym kroku dodasz wiązanie pionowe od góry Buttondo dołu TextView. Spowoduje to ustawienie Buttonponiżej TextView.

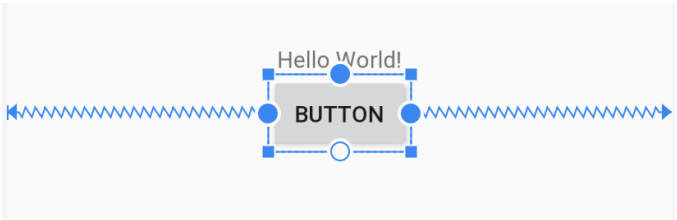
1. W widoku **Projekt** , przy górnej krawędzi Button, naciśnij i przytrzymaj białe kółko z niebieską ramką. Przeciągnij wskaźnik, a strzałka podąży za wskaźnikiem. Zwolnij, gdy dojdziesz do dolnej krawędzi „Hello World” TextView. Ustanawia to ograniczenie układu, a Buttonslajdy przesuwają się tuż pod TextView.



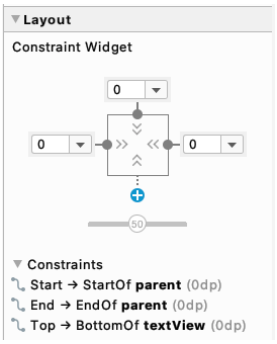
1. Spójrz na **Atrybuty** po prawej stronie **Edytora układu** .
2. W **widżecie Ograniczenie** zwróć uwagę na nowe ograniczenie układu, które jest ustawione na dole TextView, **Top → BottomOf textView (0dp)** . **(0dp)** oznacza, że istnieje margines równy 0. Występuje również błąd dotyczący brakujących ograniczeń poziomych.



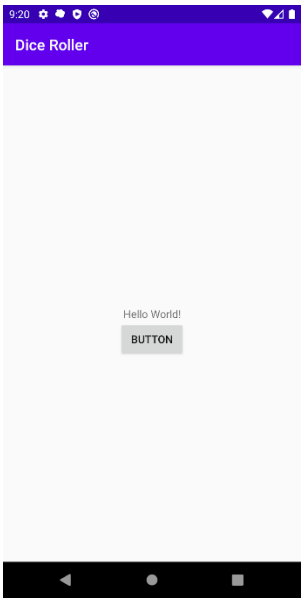
1. Dodaj wiązanie poziome od lewej strony `Button`do lewej strony rodzica `ConstraintLayout`.
2. Powtórz po prawej stronie, łącząc prawą krawędź z `Button`prawą krawędzią `ConstraintLayout`. Wynik powinien wyglądać tak:



1. Przy `Button`wciąż zaznaczonym **widżecie Ograniczenie** powinien wyglądać tak. Zwróć uwagę na dwa dodatkowe ograniczenia, które zostały dodane: **Start → StartOf parent (0dp)** i **End → EndOf parent (0dp)** . Oznacza to, że `Button`jest wyśrodkowany poziomo w swoim rodzicu, `ConstraintLayout`.



1. Uruchom aplikację. Powinno to wyglądać jak na poniższym zrzucie ekranu. Możesz kliknąć `Button`, ale to jeszcze nic nie robi. Idźmy dalej!

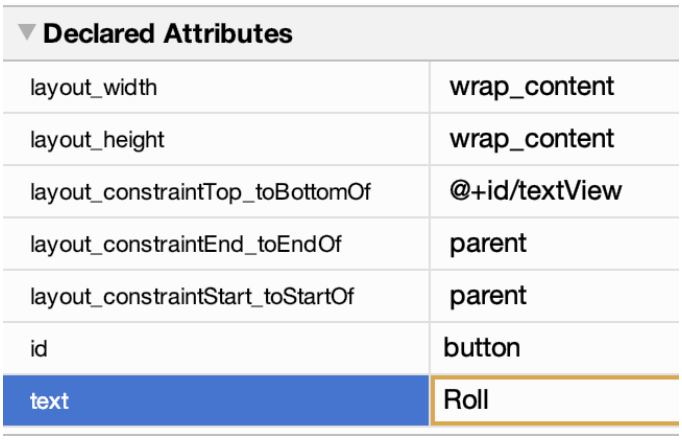


Zmień tekst przycisku

Zamierzasz wprowadzić jeszcze kilka zmian interfejsu użytkownika w **Edytorze układu** .

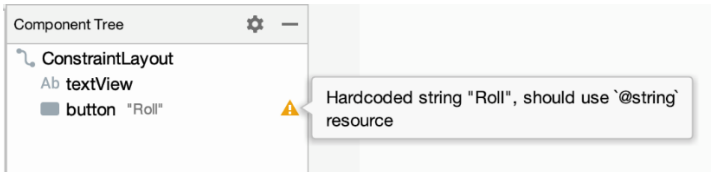
Zamiast `Button`wyświetlać etykietę „Przycisk”, zmień ją na coś, co wskazuje, co będzie robił przycisk: „Roll”.

1. W **Edytorze układu** , po `Button`wybraniu, przejdź do **Atrybuty** , zmień **tekst** na **Rolka** i naciśnij klawisz `Enter`(`Return`na Macu).

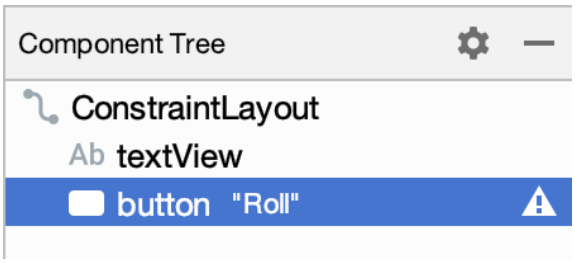


1. W **drzewie komponentów** obok `Button`. Jeśli najedziesz kursorem na trójkąt, pojawi się komunikat. Android Studio wykrył *zakodowany ciąg znaków* („Roll”) w kodzie aplikacji i sugeruje zamiast tego użycie *zasobu ciągu* .

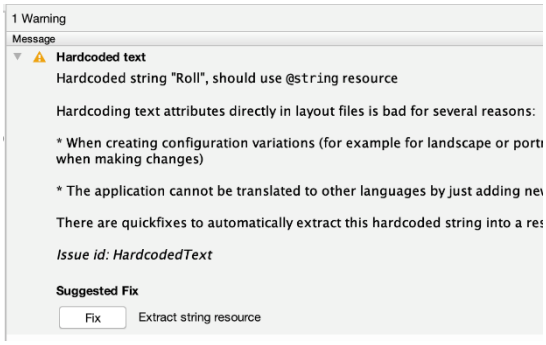
Posiadanie zakodowanego ciągu znaków oznacza, że aplikacja będzie trudniejsza do przetłumaczenia na inne języki i trudniej będzie ponownie użyć ciągów w różnych częściach aplikacji. Na szczęście Android Studio ma dla Ciebie automatyczną poprawkę.



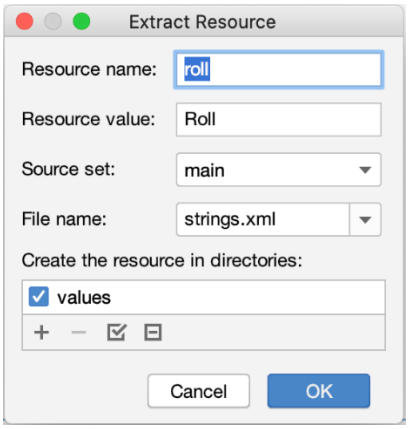
1. W **drzewie komponentów** kliknij pomarańczowy trójkąt.



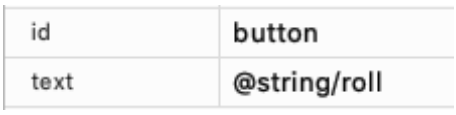
Zostanie otwarty pełny komunikat ostrzegawczy.



1. U dołu wiadomości w obszarze **Sugerowana naprawa** kliknij przycisk **Napraw** . (Może być konieczne przewinięcie w dół).
2. Otworzy się okno dialogowe **Wyodrębnij zasób .** Aby wyodrębnić ciąg znaków, należy wziąć tekst „Roll” i utworzyć zasób ciągu o nazwie `roll`in `strings.xml`(**app > res > values > strings.xml**). Wartości domyślne są poprawne, więc kliknij **OK** .



1. Zauważ, że w **Attributes** , atrybut **tekstowy** dla `Button`now mówi `@string/roll`, odwołując się do właśnie utworzonego zasobu.



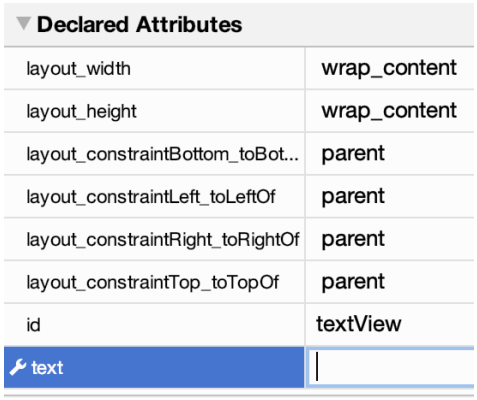
W widoku **Projekt**`Button` powinien nadal wyświetlać się napis **Roll** on it.

Stylizuj widok tekstu

„Witaj świecie!” tekst jest dość mały, a wiadomość nie pasuje do Twojej aplikacji. W tym kroku zastąpisz małe „Hello, World!” wiadomość z numerem, aby pokazać wyrzuconą wartość i powiększyć czcionkę, aby była łatwiejsza do zobaczenia.

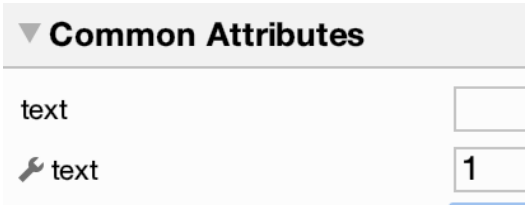
1. W **Edytorze projektu** wybierz `TextView`tak, aby jego atrybuty pojawiały się w oknie **Atrybuty** .
2. Zmień **textSize** na **36sp**`TextView` , aby był duży i łatwy do odczytania. Może być konieczne przewinięcie, aby znaleźć **textSize** .

1. Wyczyść atrybut **tekstowy**`TextView` . Nie musisz niczego wyświetlać, `TextView`dopóki użytkownik nie rzuci kostką.



Jednak `TextView`podczas edytowania układu i kodu aplikacji bardzo pomocne jest wyświetlanie tekstu w aplikacji. W tym celu możesz dodać tekst do tego `TextView`, który jest widoczny tylko w podglądzie układu, ale nie podczas działania aplikacji.

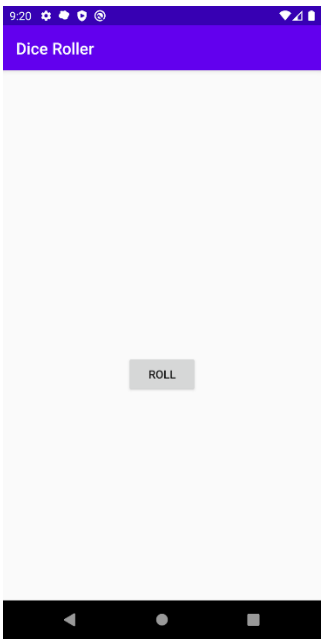
1. Wybierz `TextView`w **drzewie komponentów** .
2. W sekcji **Wspólne atrybuty** znajdź atrybut **tekstowy** , a pod nim inny atrybut **tekstowy** z ikoną narzędzia. **Atrybut tekstowy** jest tym, co będzie wyświetlane użytkownikowi, gdy aplikacja jest uruchomiona. **Atrybut tekstowy** z ikoną narzędzia to atrybut „tekst narzędzi”, który jest przeznaczony tylko dla Ciebie jako programisty .
3. Ustaw tekst narzędzi na „1” w `TextView`(aby udawać, że rzucasz kostką 1). „1” pojawi się tylko w **edytorze projektu** w Android Studio, ale nie pojawi się, gdy uruchomisz aplikację na rzeczywistym urządzeniu lub emulatorze.



Pamiętaj, że ponieważ ten tekst jest wyświetlany tylko przez programistów aplikacji, nie musisz tworzyć dla niego zasobu ciągu.

1. Spójrz na swoją aplikację w podglądzie. Pokazuje się „1”.

1. Uruchom swoją aplikację. Tak wygląda aplikacja uruchomiona na emulatorze. „1” nie jest wyświetlane. To jest prawidłowe zachowanie.



Świetnie, skończyłeś ze zmianami układu!

Masz aplikację z przyciskiem, ale jeśli go dotkniesz, nic się nie dzieje. Aby to zmienić, musisz napisać kod Kotlina, który rzuca kostką i aktualizuje ekran po naciśnięciu przycisku.

Aby wprowadzić tę zmianę, musisz trochę lepiej zrozumieć strukturę aplikacji na Androida.

4. Wprowadzenie do zajęć

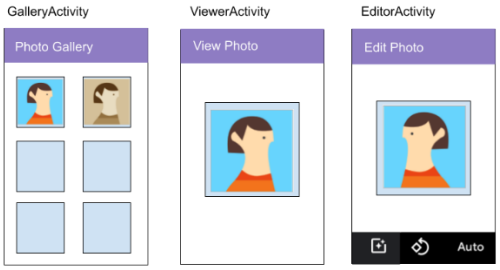
An `Activity` zapewnia okno, w którym aplikacja rysuje swój interfejs użytkownika. Zazwyczaj an `Activity` zajmuje cały ekran aplikacji do biegania. Każda aplikacja ma co najmniej jedną aktywność. Działanie najwyższego poziomu lub pierwsze jest często nazywane działaniem `MainActivity` i jest dostarczane przez szablon projektu. Na przykład, gdy użytkownik przewinie listę aplikacji na swoim urządzeniu i dotknie ikony aplikacji „Dice Roller", system Android uruchomi `MainActivity` aplikację.

W swoim `MainActivity` kodzie musisz podać szczegóły dotyczące `Activity` układu i sposobu, w jaki użytkownik powinien z nim korzystać.

- W aplikacji Kartka urodzinowa jest taka `Activity`, która wyświetla wiadomość i obraz o urodzinach.
- W aplikacji Dice Roller jest taka `Activity`, która wyświetla układ `TextView` i `Button` układ, który właśnie zbudowałeś.

W przypadku bardziej skomplikowanych aplikacji może być wiele ekranów i więcej niż jeden `Activity`. Każdy `Activity` ma określony cel.

Na przykład w aplikacji galerii zdjęć możesz mieć jedną `Activity` do wyświetlania siatki zdjęć, drugą `Activity` do przeglądania pojedynczego zdjęcia, a trzecią `Activity` do edycji pojedynczego zdjęcia.



Otwórz plik MainActivity.kt

Dodasz kod, aby odpowiedzieć na dotknięcie przycisku w `MainActivity`. Aby zrobić to poprawnie, musisz dowiedzieć się więcej o `MainActivity` kodzie, który jest już w Twojej aplikacji.

1. Przejdź do pliku i otwórz `MainActivity.kt` go (**app > java > com.example.diceroller > MainActivity.kt**). Poniżej znajduje się to, co powinieneś zobaczyć. Jeśli widzisz `import...`, kliknij , `...` aby rozwinąć importy.

```
package com.example.diceroller

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
```

```
class MainActivity : AppCompatActivity() {

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
  }
}
```

Nie musisz rozumieć każdego słowa z powyższego kodu, ale musisz mieć ogólne pojęcie o tym, co robi. Im więcej będziesz pracował z kodem Androida, tym bardziej się on stanie i tym lepiej go zrozumiesz.

1. Spójrz na kod Kotlin dla `MainActivity`klasy, zidentyfikowany przez słowo kluczowe `class`, a następnie nazwę.

```
class MainActivity : AppCompatActivity() {
  ...
}
```

1. Zauważ, że nie ma `main()`funkcji w twoim `MainActivity`.

Wcześniej dowiedziałeś się, że każdy program Kotlina musi mieć `main()`funkcję. Aplikacje na Androida działają inaczej. Zamiast wywoływać `main()`funkcję, system Android wywołuje `onCreate()`metodę Twojej `MainActivity`, gdy Twoja aplikacja jest otwierana po raz pierwszy.

1. Znajdź `onCreate()`metodę, która wygląda jak poniższy kod.

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContentView(R.layout.activity_main)
}
```

Dowiesz się o `override`tym w późniejszych ćwiczeniach z programowania (więc nie martw się tym na razie). Pozostała część `onCreate()`metody konfiguruje metodę, `MainActivity`używając kodu z importów i ustawiając układ początkowy za pomocą `setContentView()`.

1. Zwróć uwagę na linie zaczynające się od `import`.

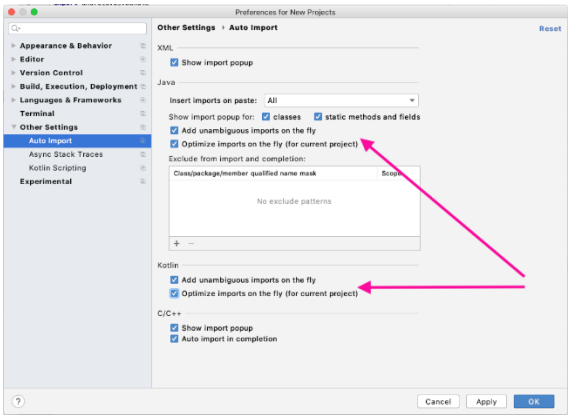
Android udostępnia *strukturę* wielu klas, które ułatwiają pisanie aplikacji na Androida, ale musi dokładnie wiedzieć, o którą klasę chodzi. Za pomocą `import`instrukcji można określić, która klasa w strukturze ma być używana w kodzie. Na przykład `Button`klasa jest zdefiniowana w `android.widget.Button`.

Włącz automatyczne importowanie

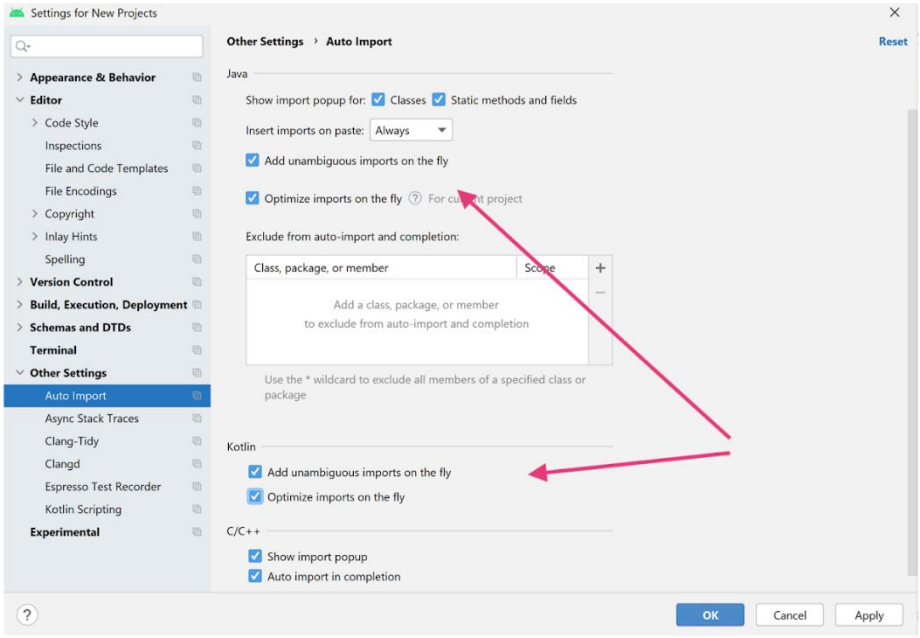
`import`Pamiętanie o dodawaniu instrukcji, gdy używasz większej liczby klas, może wymagać dużo pracy . Na szczęście Android Studio pomaga wybrać prawidłowe importy, gdy korzystasz z klas

dostarczonych przez innych. W tym kroku skonfigurujesz Android Studio tak, aby automatycznie dodawał importy, kiedy to możliwe, i automatycznie usuwał nieużywane importy z kodu.

W systemie macOS otwórz ustawienia, przechodząc do **Plik > Ustawienia nowego projektu > Preferencje dla nowych projektów...** Rozwiń **Inne ustawienia > Importuj automatycznie** . W sekcjach **Java** i **Kotlin** upewnij się, że zaznaczone są **Dodaj jednoznaczne importy w locie** i **Optymalizuj importy w locie (dla bieżącego projektu) .** Zwróć uwagę, że w każdej sekcji znajdują się dwa pola wyboru. Zapisz zmiany i zamknij ustawienia, naciskając **OK** .



W systemie Windows otwórz ustawienia, przechodząc do **Plik > Ustawienia > Edytor > Ogólne > Importuj automatycznie** . W sekcjach **Java** i **Kotlin** upewnij się, że zaznaczone są **Dodaj jednoznaczne importy w locie** i **Optymalizuj importy w locie (dla bieżącego projektu) .** Zwróć uwagę, że w każdej sekcji znajdują się dwa pola wyboru. Zapisz zmiany i zamknij ustawienia, naciskając **OK** .



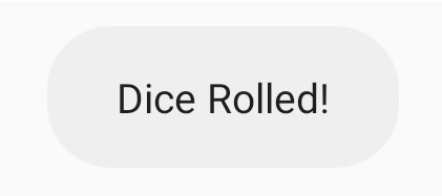
Jednoznaczne ustawienia **importu** informują Android Studio, aby automatycznie dodał instrukcję importu, o ile może określić, której użyć. Ustawienia **optymalizacji importów** informują Android Studio, aby usunąć wszelkie importy, które nie są używane przez Twój kod.

5. Spraw, aby przycisk był interaktywny

Teraz, gdy wiesz trochę więcej o `MainActivity`, zmodyfikujesz aplikację tak, aby kliknięcie `Button`robiło coś na ekranie.

Wyświetlaj komunikat po kliknięciu przycisku

W tym kroku określisz, że po kliknięciu przycisku na dole ekranu pojawi się krótki komunikat.



1. Dodaj następujący kod do `onCreate()`metody po `setContentView()`wywołaniu. Metoda `findViewById()`znajduje `Button`w układzie. `R.id.button`jest identyfikatorem zasobu dla `Button`, który jest dla niego unikalnym identyfikatorem. Kod zapisuje *odwołanie* do `Button`obiektu w zmiennej o nazwie `rollButton`, a nie `Button`sam obiekt.

```
val rollButton: Button = findViewById(R.id.button)
```

Uwaga: Android automatycznie przypisuje numery identyfikacyjne do zasobów w Twojej aplikacji. Na przykład przycisk **Roll** ma identyfikator zasobu, a ciąg tekstu przycisku ma również identyfikator zasobu. Identyfikatory zasobów mają postać `R.<type>.<name>`; na przykład `R.string.roll`. W przypadku `View`identyfikatorów `<type>`jest to `id`na przykład `R.id.button`.

Kod zapisuje odwołanie do `Button`obiektu w zmiennej o nazwie `rollButton`, a nie `Button`sam obiekt.

Ważne: Gdy Kotlin przypisuje obiekt do zmiennej, nie kopiuje za każdym razem całego obiektu, zapisuje *odwołanie* do obiektu. Możesz pomyśleć o odnośniku podobnym do krajowego numeru identyfikacyjnego; liczba odnosi się do osoby, ale nie jest samą osobą. Kiedy kopiujesz numer, nie kopiujesz osoby.

Metoda `onCreate()`powinna teraz wyglądać tak.

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContentView(R.layout.activity_main)

  val rollButton: Button = findViewById(R.id.button)
}
```

1. Sprawdź, czy Android Studio automatycznie dodało `import`oświadczenie dla `Button`. Zauważ, że są teraz 3 instrukcje importu.

```
import android.os.Bundle
import android.widget.Button
```

```
import androidx.appcompat.app.AppCompatActivity
```

Uwaga: jeśli włączenie automatycznego importowania nie zadziałało, `Button`zostanie podświetlone na czerwono. Możesz ręcznie dodać poprawny import, umieszczając kursor tekstowy w wyrazie `Button`, a następnie naciskając `Alt+Enter`(`Option+Enter`na Macu).

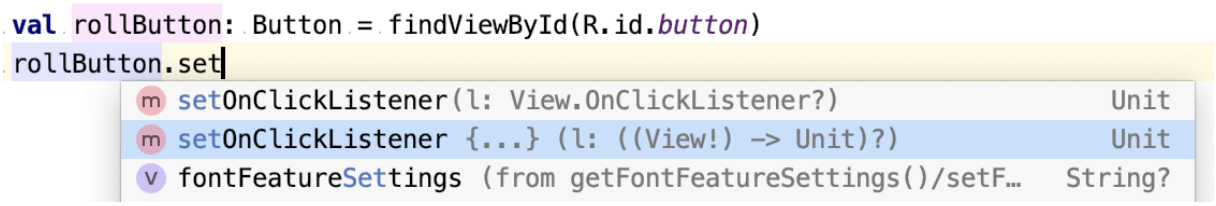
Następnie musisz powiązać kod z `Button`, aby kod mógł zostać wykonany po `Button`dotknięciu. Odbiornik *kliknięć* to kod określający, co zrobić, gdy nastąpi dotknięcie lub kliknięcie. Możesz myśleć o tym jako o kodzie, który po prostu siedzi, „nasłuchując", jak użytkownik klika, w tym przypadku, na `Button`.

1. Użyj `rollButton`obiektu i ustaw na nim detektor kliknięcia, wywołując `setOnClickListener()`metodę. Zamiast nawiasów po nazwie metody, w rzeczywistości będziesz używać nawiasów klamrowych po nazwie metody. Jest to specjalna składnia do deklarowania Lambda , o której dowiesz się więcej w przyszłych ćwiczeniach z programowania.

Na razie musisz wiedzieć, że w nawiasach klamrowych umieszczasz instrukcje dotyczące tego, co powinno się stać po naciśnięciu przycisku. Twoja aplikacja wyświetli Toast, który jest krótką wiadomością w następnym kroku.

```
rollButton.setOnClickListener {
}
```

Podczas pisania Android Studio może wyświetlać wiele sugestii. W tym przypadku wybierz opcję **setOnClickListener {...}** .



W nawiasach klamrowych umieszczasz instrukcje dotyczące tego, co powinno się stać po naciśnięciu przycisku. Na razie Twoja aplikacja będzie wyświetlać ikonę Toast, która jest krótką wiadomością wyświetlaną użytkownikowi.

1. Utwórz `Toast`z tekstem `"Dice Rolled!"`dzwoniąc `Toast.makeText()`.

```
val toast = Toast.makeText(this, "Dice Rolled!", Toast.LENGTH_SHORT)
```

1. Następnie powiedz, `Toast`aby wyświetlał się, wywołując `show()`metodę.

```
toast.show()
```

Tak wygląda twoja zaktualizowana `MainActivity`klasa; oświadczenia `package`i `import`nadal znajdują się na początku pliku:

```
class MainActivity : AppCompatActivity() {
```

```
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val rollButton: Button = findViewById(R.id.button)
    rollButton.setOnClickListener {
      val toast = Toast.makeText(this, "Dice Rolled!", Toast.LENGTH_SHORT)
      toast.show()
    }
  }
}
```

Możesz połączyć dwie linie w odbiorniku kliknięć w jedną linię bez zmiennej. Jest to typowy wzorzec, który możesz znaleźć w innym kodzie.

```
Toast.makeText(this, "Dice Rolled!", Toast.LENGTH_SHORT).show()
```

1. Uruchom aplikację i kliknij przycisk **Rolka** . Na dole ekranu powinien pojawić się wyskakujący komunikat i po chwili zniknąć.

Hurra! Kliknięcie przycisku spowodowało wyświetlenie komunikatu! Po raz pierwszy napisałeś kod Kotlin na Androida!

Zaktualizuj TextView po kliknięciu przycisku

Zamiast wyświetlać tymczasową `Toast`wiadomość, napiszesz kod aktualizujący `TextView`ekran po kliknięciu przycisku **Roll .**

1. Wróć do `activity_main.xml`(**app > res > layout > activity_main.xml**)

2. Kliknij na `TextView`.
3. Zauważ, że **identyfikator** to **textView** .

1. Otwórz `MainActivity.kt`(**app > java > com.example.diceroller > MainActivity.kt**)
2. Usuń wiersze kodu, które tworzą i pokazują `Toast`.

```
rollButton.setOnClickListener {

}
```

1. W ich miejsce utwórz nową zmienną o nazwie `resultTextView`do przechowywania `TextView`.
2. Użyj `findViewById()`, aby znaleźć `textView`w układzie za pomocą jego identyfikatora i przechowywać odniesienie do niego.

```
val resultTextView: TextView = findViewById(R.id.textView)
```

1. Ustaw tekst na `resultTextView`„6” w cudzysłowie.

```
resultTextView.text = "6"
```

Jest to podobne do tego, co zrobiłeś, ustawiając **tekst** w **Attributes** , ale teraz jest on w twoim kodzie, więc tekst musi być wewnątrz podwójnych cudzysłowów. Ustawienie tego wyraźnie oznacza, że na razie `TextView`zawsze wyświetla 6. Dodasz kod, aby rzucić kostką i pokazywać różne wartości w następnym zadaniu.

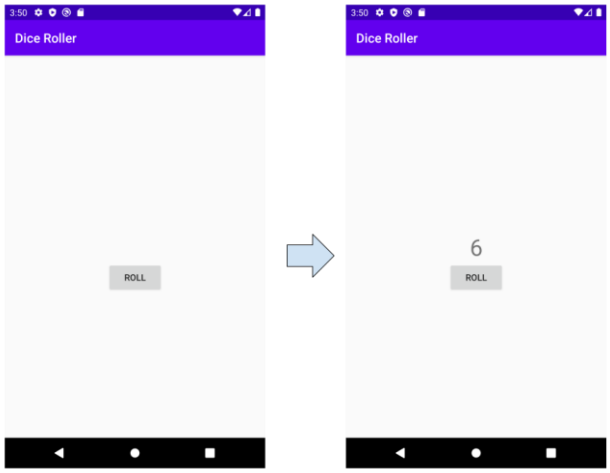
Tak `MainActivity`powinna wyglądać klasa:

```
class MainActivity : AppCompatActivity() {

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val rollButton: Button = findViewById(R.id.button)
    rollButton.setOnClickListener {
      val resultTextView: TextView = findViewById(R.id.textView)
      resultTextView.text = "6"
    }
  }
}
```

1. Uruchom aplikację. Naciśnij przycisk. Powinien zaktualizować `TextView`do „6”.



6. Dodaj logikę rzutu kostką

Jedyne, czego brakuje, to rzucanie kostką. Możesz ponownie użyć `Dice`klasy z poprzedniego ćwiczenia z programowania, które obsługuje logikę rzucania kostką.

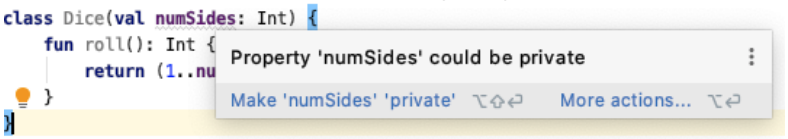
Dodaj klasę Dice

1. Po ostatnim nawiasie klamrowym w `MainActivity`klasie utwórz `Dice`klasę za pomocą `roll()`metody.

```
class Dice(val numSides: Int) {

  fun roll(): Int {
    return (1..numSides).random()
  }
}
```

1. Zwróć uwagę, że Android Studio może być podkreślony `numSides`szarą falistą linią. (To może chwilę potrwać, zanim się pojawi).
2. Umieść wskaźnik myszy nad `numSides`, a pojawi się wyskakujące okienko z informacją, że **właściwość „numSides" może być prywatna** .



Oznaczenie `numSides`jako `private`sprawi, że będzie dostępne tylko w ramach `Dice`klasy. Ponieważ jedyny kod, który będzie używany, `numSides`znajduje się wewnątrz `Dice`klasy, można użyć tego argumentu `private`dla `Dice`klasy. W następnej części dowiesz się więcej o `private`kontraście `public`ze zmiennymi.

1. Śmiało i wprowadź sugerowaną poprawkę z Android Studio, klikając **Ustaw 'numSides' 'private'** .

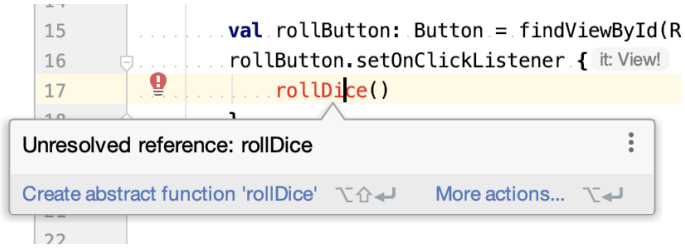
Utwórz metodę rollDice()

Teraz, gdy dodałeś `Dice`zajęcia do swojej aplikacji, zaktualizujesz je, `MainActivity`aby z nich korzystać. Aby lepiej zorganizować swój kod, umieść całą logikę dotyczącą rzucania kostką w jedną funkcję.

1. Zastąp kod w odbiorniku kliknięć, który ustawia tekst na „6”, wywołaniem `rollDice()`.

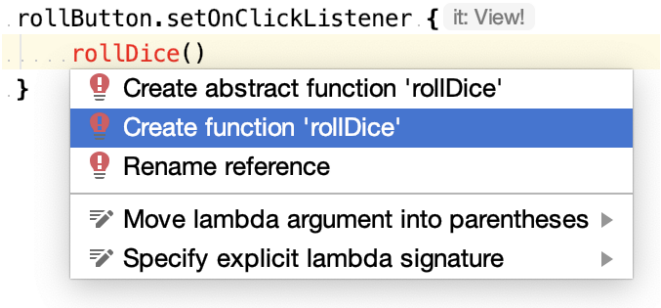
```
rollButton.setOnClickListener {
  rollDice()
}
```

1. Ponieważ `rollDice()`nie jest jeszcze zdefiniowany, Android Studio oznacza błąd i wyświetla `rollDice()`się na czerwono.
2. Jeśli najedziesz kursorem na `rollDice()`, Android Studio wyświetli problem i kilka możliwych rozwiązań.



1. Kliknij **Więcej akcji...** , co spowoduje wyświetlenie menu. Android Studio oferuje więcej pracy dla Ciebie!

Wskazówka: jeśli trudno jest najechać kursorem, a następnie kliknąć **Więcej akcji...** , możesz kliknąć `rollDice()`i nacisnąć `Alt+Enter`(`Option+Enter`na Macu), aby wyświetlić menu.



1. Wybierz **Utwórz funkcję 'rollDice'** . Android Studio tworzy pustą definicję funkcji wewnątrz `MainActivity`.

```
private fun rollDice() {
  TODO("Not yet implemented")
}
```

Utwórz nową instancję obiektu Dice

W tym kroku `rollDice()`utworzysz metodę i rzucisz kostką, a następnie wyświetlisz wynik w `TextView`.

1. Wewnątrz `rollDice()`usuń `TODO()`połączenie.

2. Dodaj kod, aby utworzyć kostkę z 6 stronami.

```
val dice = Dice(6)
```

1. Rzuć kostką, wywołując `roll()`metodę, i zapisz wynik w zmiennej o nazwie `diceRoll`.

```
val diceRoll = dice.roll()
```

1. Znajdź `TextView`dzwoniąc `findViewById()`.

```
val resultTextView: TextView = findViewById(R.id.textView)
```

Zmienna `diceRoll`jest liczbą, ale `TextView`używa tekstu. Możesz użyć `toString()`metody on, `diceRoll`aby przekonwertować ją na ciąg.

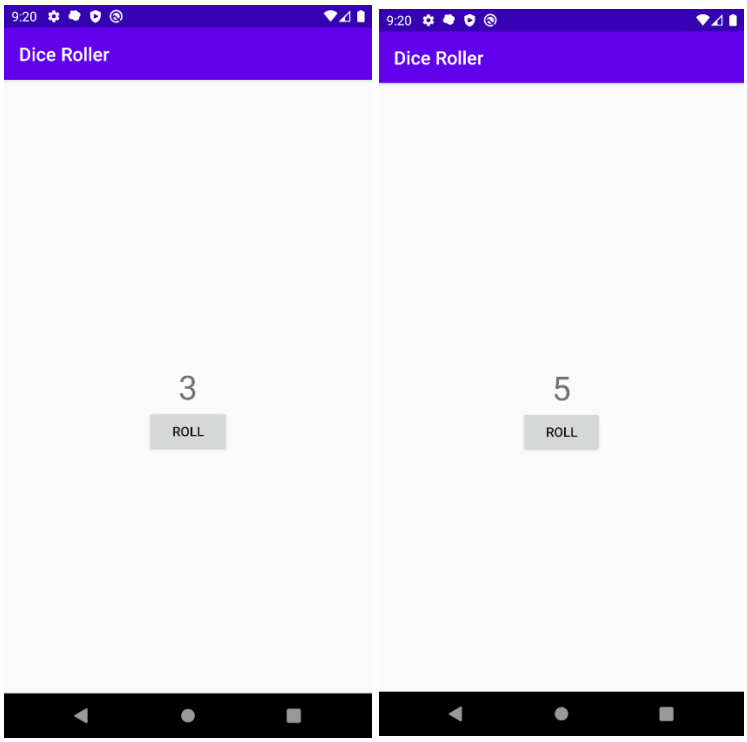
1. Przekonwertuj `diceRoll`na ciąg i użyj go, aby zaktualizować tekst `resultTextView`.

```
resultTextView.text = diceRoll.toString()
```

Tak `rollDice()`wygląda metoda:

```
private fun rollDice() {
   val dice = Dice(6)
   val diceRoll = dice.roll()
   val resultTextView: TextView = findViewById(R.id.textView)
   resultTextView.text = diceRoll.toString()
}
```

1. Uruchom swoją aplikację. Wynik kości powinien zmienić się na inne wartości poza 6! Ponieważ jest to liczba losowa od 1 do 6, czasami może pojawić się również wartość 6.



Hurra, bujasz!

7. Zastosuj dobre praktyki kodowania

To normalne, że Twój kod wygląda trochę niechlujnie po dostrojeniu części tu i tam, aby Twoja aplikacja działała. Ale zanim odejdziesz od swojego kodu, powinieneś wykonać kilka prostych zadań porządkowych. Wtedy aplikacja będzie w dobrym stanie i będzie łatwiejsza w utrzymaniu.

Te nawyki praktykują profesjonalni programiści Androida podczas pisania kodu.

Przewodnik po stylu Androida

Podczas pracy w zespołach członkowie zespołu mogą pisać kod w podobny sposób, dzięki czemu istnieje pewna spójność w całym kodzie. Dlatego Android ma Przewodnik po stylach, jak pisać kod Androida — konwencje nazewnictwa, formatowanie i inne dobre praktyki, których należy przestrzegać. Podczas pisania kodu na Androida przestrzegaj tych wytycznych: Przewodnik po stylu Kotlin dla programistów aplikacji na Androida .

Poniżej przedstawiamy kilka sposobów, w jakie możesz stosować się do przewodnika po stylu.

Wyczyść swój kod

Skondensuj swój kod

Możesz uczynić swój kod bardziej zwięzłym poprzez skondensowanie kodu w krótszą liczbę wierszy. Na przykład tutaj jest kod, który ustawia detektor kliknięć na `Button`.

```
rollButton.setOnClickListener {
   rollDice()
}
```

Ponieważ instrukcje dla odbiornika kliknięć mają tylko 1 linię, możesz skondensować `rollDice()`wywołanie metody i nawiasy klamrowe w jednym wierszu. Tak to wygląda. Jedna linia zamiast trzech!

```
rollButton.setOnClickListener { rollDice() }
```

8. Kod rozwiązania

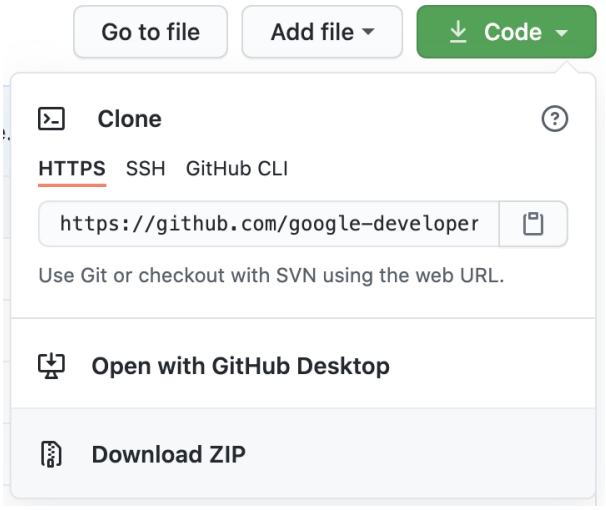
Kod rozwiązania dla tego ćwiczenia programowania znajduje się w projekcie i module pokazanym poniżej.

URL kodu rozwiązania: https://github.com/google-developer-training/android-basics-kotlin-create-dice-roller-with-button-app-solution

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

Pobierz kod

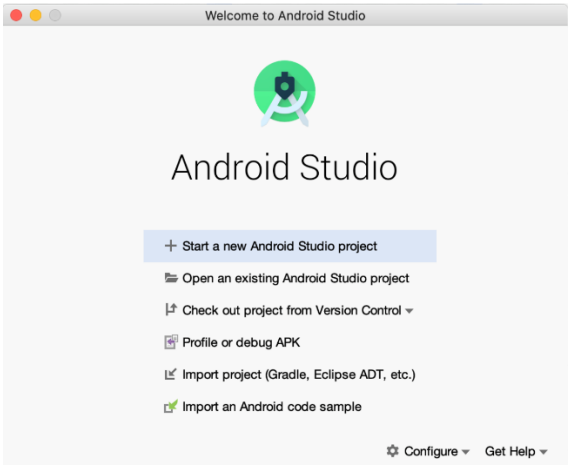
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



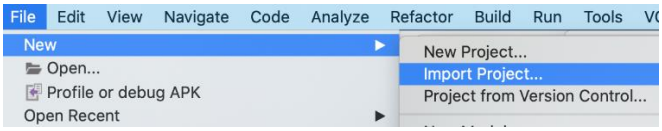
1. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
2. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
3. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego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



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

9. Podsumowanie

- Dodaj znak `Button`w aplikacji na Androida za pomocą **Edytora układu** .
- Zmodyfikuj `MainActivity.kt`klasę, aby dodać interaktywne zachowanie do aplikacji.
- Wyświetl `Toast`wiadomość jako tymczasowe rozwiązanie, aby sprawdzić, czy jesteś na dobrej drodze.
- Ustaw odbiornik po kliknięciu do `Button`użycia `setOnClickListener()`, aby dodać zachowanie po `Button`kliknięciu.
- Gdy aplikacja jest uruchomiona, możesz zaktualizować ekran, wywołując metody w `TextView`, `Button`lub innych elementach interfejsu użytkownika w układzie.

- Skomentuj swój kod, aby pomóc innym osobom, które go czytają, zrozumieć, jakie było Twoje podejście.
- Sformatuj kod i wyczyść kod.

10. Dowiedz się więcej

- Słownictwo dla Androida Podstawy w Kotlin
- `Button`klasa
- `Toast`klasa
- `TextView`klasa
- Przewodnik po stylu Kotlin dla programistów Androida

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:

1. Dodaj kolejną kostkę do aplikacji. Kliknięcie przycisku **Rzuć** powinno rzucić 2 kostkami. Wyniki powinny być wyświetlane w 2 różnych `TextViews`na ekranie.

Sprawdź swoją pracę:

Twoja gotowa aplikacja powinna działać bez błędów i pokazywać dwie kostki w aplikacji.

Dodaj zachowanie warunkowe w Kotlin

1. Zanim zaczniesz

W tym laboratorium kodów Kotlin stworzysz kolejną grę w kości, Lucky Dice Roll, próbując wyrzucić szczęśliwą liczbę. Twój program ustawi szczęśliwą liczbę i rzuci kostką. Następnie sprawdzasz rzut pod kątem szczęśliwej liczby i wyświetlasz odpowiednią wiadomość na wyjściu. Aby to osiągnąć, nauczysz się porównywać wartości i podejmować różne decyzje w swoim programie Kotlin.

Aby pomóc Ci skupić się na koncepcjach programowania bez martwienia się o interfejs użytkownika aplikacji, użyjesz narzędzia programistycznego Kotlin opartego na przeglądarce i prześlesz swoje wyniki do konsoli.

Warunki wstępne

- Jak otwierać, edytować i uruchamiać kod na https://developer.android.com/training/kotlinplayground
- Możliwość tworzenia i uruchamiania programu Kotlin, który wykorzystuje zmienne, funkcje z argumentami, klasy z metodami i wyświetla wynik na konsoli.

Czego się nauczysz

- Jak używać `if`i `else`oświadczenia.
- Jak porównywać wartości za pomocą operatorów, takich jak większa niż (`>`), mniejsza niż (`<`) i równa (`==`).
- Jak korzystać z `when`instrukcji, aby wybrać opcję w oparciu o daną wartość.
- Czym `Boolean`jest typ danych i jak używać jego `true`i `false`wartości do podejmowania decyzji.

Co zbudujesz

- Oparta na Kotlinie gra w kości, Lucky Dice Roll, która pozwala określić szczęśliwą liczbę. Gracz wygra, jeśli wyrzuci szczęśliwą liczbę.

Czego potrzebujesz

- Komputer z połączeniem internetowym.

2. Podejmowanie decyzji w Twoim kodzie

Twój program Lucky Dice Roller musi określić, czy użytkownik wyrzucił szczęśliwą liczbę i otrzyma gratulacje, czy zamiast tego otrzyma wiadomość, aby spróbować ponownie.

Jako programista aplikacji musisz podejmować decyzje o tym, jak aplikacja powinna się zachowywać, i tworzyć różne wyniki dla użytkowników.

Jeśli tworzysz aplikację zakupową, możesz wyświetlać różne ekrany w zależności od opcji dostawy wybranych przez użytkownika. W przypadku quizu wyświetlałeś różne ekrany w zależności od tego, czy odpowiedź gracza jest poprawna. W zależności od aplikacji może istnieć wiele możliwych wyników, które będziesz chciał uwzględnić w swoim kodzie.

W programie Lucky Dice Roller aplikacja powinna obsługiwać różne przypadki, takie jak:

- **Jeśli** rzut kostką jest szczęśliwą liczbą, wyświetl wiadomość z gratulacjami!
- **W przeciwnym razie, jeśli** rzut kostką nie jest szczęśliwą liczbą, wyświetl komunikat, aby spróbować ponownie.

Aby dodać tę logikę do swojego kodu, użyj specjalnych słów kluczowych Kotlin, takich jak `if`i `else`i `when`.

Spójrzmy na kilka przykładów.

Użyj **if**instrukcji, aby ustawić warunek, który jest spełniony

1. Sprawdź poniższy kod. Czy możesz zgadnąć, jaki będzie wynik?

```
fun main() {
  val num = 5
  if (num > 4) {
    println("The variable is greater than 4")
  }
}
```

1. Skopiuj i wklej kod w edytorze programu Kotlin i uruchom program, aby obserwować wyjście.

```
The variable is greater than 4
```

Proces decyzyjny dla tego programu to:

1. Utwórz zmienną `num`i ustaw ją na`5`.
2. Jeśli to prawda, że `num`jest większe niż 4, print `"The variable is greater than 4"`.
3. We wszystkich innych sytuacjach nic nie rób.

W powyższym przykładzie `num`jest ustawiony na 5. `if`Instrukcja porównuje, czy zmienna jest większa niż 4. Ponieważ to prawda, system następnie wykonuje instrukcje w nawiasach klamrowych i wyświetla komunikat.

Przypomnienie: symbol `>`jest operatorem do porównywania dwóch wartości, niezależnie od tego, czy pierwsza wartość jest większa od drugiej wartości, i zwraca wynik `true`lub `false`. Inne popularne operatory to: `<`dla mniejszego niż, `==`dla równego, `>=`dla większego lub równego oraz `<=`dla mniejszego lub równego.

Zwróć uwagę na ogólny format `if`oświadczenia:

- Zacznij od `if`słowa kluczowego.
- Dalej z dwoma nawiasami `()`. W nawiasach znajduje się warunek. Warunkiem jest wszystko, co może być `true`lub `false`. Na przykład, czy liczba jest większa od innej.
- Śledź z dwoma nawiasami klamrowymi `{}`. Wewnątrz nawiasów klamrowych umieszczasz kod do wykonania, jeśli warunek jest `true`.

```
if (condition-is-true) {
  execute-this-code
}
```

Typ danych logicznych dla wartości prawda i fałsz:

- Program sprawdzający, czy warunek jest spełniony, nazywa się „oceną warunku”. Wynikiem oceny warunku jest `true`to, czy jest on spełniony, czy `false`nie. Deweloperzy mówią również: „stan ocenia się na `true`” lub „stan ocenia się na `false`„.
- Warunki mogą oceniać tylko do `true`lub `false`.
- Tak jak istnieje typ danych `Int`dla liczb całkowitych i `IntRange`dla zakresów, istnieje typ danych dla `true`i `false`, o nazwie `Boolean`. `Boolean`W dalszej części tego kursu napotkasz zmienne typu.

Użyj **if**instrukcji, aby ustawić warunek, który nie jest spełniony

1. Zmień wartość `num`na 3, jak pokazano poniżej. Czego się spodziewasz, jeśli uruchomisz ten kod?

```
fun main() {
  val num = 3
  if (num > 4) {
    println("The variable is greater than 4")
  }
}
```

1. Skopiuj i wklej kod w edytorze programu Kotlin i uruchom program, aby obserwować puste wyjście.

Przy `num`ustawieniu na 3 nic nie jest drukowane, ponieważ wartość `num`jest mniejsza niż 4. Zatem warunek, który `num`jest większy niż 4, to `false`, a kod między nawiasami klamrowymi nie jest wykonywany i nic nie jest drukowane.

Użyj **else**, aby stworzyć alternatywę dla nieudanych warunków

Zamiast nic nie robić, możesz zaoferować użytkownikom alternatywę, gdy warunek nie zostanie spełniony. Podobnie jak w przypadku zwykłego języka, możesz to zrobić za pomocą `else`instrukcji.

1. Dodaj `else`oświadczenie, aby wydrukować komunikat, gdy `num`nie jest większy niż 4, jak pokazano poniżej. Czego się spodziewasz, jeśli uruchomisz ten kod?

```
fun main() {
  val num = 3
  if (num > 4) {
    println("The variable is greater than 4")
  } else {
    println("The variable is less than 4")
  }
}
```

1. Skopiuj i wklej kod w edytorze programu Kotlin i uruchom program, aby obserwować wyjście.

```
The variable is less than 4
```

1. Zauważ, że jeśli `num`ma wartość 3, program wypisuje komunikat „ `The variable is less than 4`” powiązany z `else`instrukcją, ponieważ `num`nie jest większy niż 4.
2. Zmień `num`na 5 i uruchom program ponownie. Teraz prawdą jest, że `num`jest większe niż 4 i program wypisuje " `The variable is greater than 4`".
3. Zmień `num`na 4 i uruchom swój program. Teraz 4 nie jest większe niż 4, a program wypisuje „ `The variable is less than 4`”.

Podczas gdy " `The variable is less than 4`" jest poprawnym wynikiem dla warunków, które ustawiłeś w kodzie, to wydrukowane oświadczenie nie jest dokładne, ponieważ 4 jest nie mniejsze niż 4. Możesz dodać kolejny warunek, który sprawdza trzecią możliwość, czy `num`jest to dokładnie 4 i drukuje poprawną instrukcję, gdy ten warunek jest spełniony.

Użyj **else if**kombinacji, aby dodać alternatywne warunki

Możesz mieć więcej niż jeden warunek. Na przykład w ten sposób możesz objąć wszystkie możliwości `num`:

- **Jeśli**`num` jest większe niż 4, drukuj `"The variable is greater than 4"`.
- **W przeciwnym razie, jeśli**`num` jest równe 4, print `"The variable is equal to 4"`.
- **W przeciwnym razie** drukuj `"The variable is less than 4"`.

Są one określane jako różne przypadki w instrukcji if-else. Wymienione są 3 przypadki.

Zaktualizowany kod wygląda następująco:

```
fun main() {
  val num = 4
  if (num > 4) {
    println("The variable is greater than 4")
  } else if (num == 4) {
    println("The variable is equal to 4")
  } else {
    println("The variable is less than 4")
  }
}
```

Zwróć uwagę na następujące zmiany:

- Wartość `num`jest teraz ustawiona na 4, więc możesz przetestować nowy warunek.
- Między oryginałem `if`a `else`oświadczeniami znajduje się nowe `else if`stwierdzenie dla przypadku, gdy `num`wynosi dokładnie 4.

1. Skopiuj i wklej powyższy kod w edytorze programu Kotlin i uruchom program, aby obserwować wyjście.

```
The variable is equal to 4
```

1. Poeksperymentuj ze zmianą wartości `num`i zobacz, jak wpływa to na wynik. Zmień `num` na 2 i 6, aby zobaczyć każdy z warunków `true`.

Kontrola przepływu

Kiedy spojrzysz na powyższe instrukcje if-else, kod zostanie wykonany lub przepłynie zgodnie z warunkami. Tak więc sposób, w jaki kierujesz wykonaniem za pomocą tych warunków, nazywa się „przepływem sterowania” programu.

- Załóżmy, że Twój rzut kostką `num`wynosi 3. Program sprawdza pierwszy warunek (liczba > 4). To jest fałszywe, więc program sprawdza następny warunek (liczba == 4), który również jest fałszywy. Następnie program wykonuje kod instrukcji else, która jest ostatnią opcją.
- Jeśli liczba rzutów kośćmi wynosi 6, pierwszy warunek (liczba > 4) jest spełniony. Program wypisuje komunikat „ `The variable is greater than 4`”. Ponieważ ten warunek jest spełniony, nie trzeba sprawdzać reszty i odbywa się to za pomocą instrukcji if-else.

- Użyj kombinacji else + if, aby dodać alternatywne warunki.

3. Stwórz grę Lucky Dice Roll

W tej sekcji, korzystając z tego, czego nauczyłeś się w poprzednim zadaniu, zaktualizujesz program Dice Roller, aby sprawdzić, czy wyrzuciłeś wstępnie ustaloną szczęśliwą liczbę. Jeśli tak, wygrywasz!

Skonfiguruj swój kod startowy

Uruchamiasz Lucky Dice Roller z kodem podobnym do kodu rozwiązania z poprzedniego programu Kotlin Dice Roller. Możesz edytować `main()`funkcję w poprzednim kodzie, aby dopasować, lub możesz skopiować i wkleić poniższy kod, aby rozpocząć.

```
fun main() {
    val myFirstDice = Dice(6)
    val rollResult = myFirstDice.roll()
    println("Your ${myFirstDice.numSides} sided dice rolled ${rollResult}!")
}

class Dice (val numSides: Int) {

    fun roll(): Int {
        return (1..numSides).random()
    }
}
```

Uwaga: Za każdym razem, gdy uruchamiasz powyższy program na placu zabaw, `main()`funkcja jest wywoływana. Tworzy to nową instancję `Dice`i wywołującą `roll()`na niej metodę. Tak więc za każdym razem, gdy uruchamiasz program, wartość rzutu kostką może być inna.

Sprawdź, czy szczęśliwa liczba została wyrzucona

Najpierw stwórz szczęśliwą liczbę, a następnie porównaj rzut kostką z tą liczbą.

1. W `main()`programie usuń `println()`oświadczenie.
2. W `main()`, dodaj `val`wywołane `luckyNumber`i ustaw je na 4. Twój kod powinien wyglądać tak.

```
fun main() {
    val myFirstDice = Dice(6)
    val rollResult = myFirstDice.roll()
    val luckyNumber = 4
}
```

1. Poniżej dodaj `if`instrukcję z warunkiem w nawiasach `()`, który sprawdza, czy `rollResult`jest równe (`==`) do `luckyNumber`. Zostaw trochę miejsca między nawiasami klamrowymi `{}`, aby móc dodać więcej kodu.

```
fun main() {
   val myFirstDice = Dice(6)
   val rollResult = myFirstDice.roll()
   val luckyNumber = 4
   if (rollResult == luckyNumber) {

   }
}
```

1. Wewnątrz nawiasów klamrowych `{}`dodaj `println`oświadczenie do wydrukowania`"You win!"`

```
fun main() {
   val myFirstDice = Dice(6)
   val rollResult = myFirstDice.roll()
   val luckyNumber = 4

   if (rollResult == luckyNumber) {
      println("You win!")
   }
}
```

1. Uruchom swój program. Być może będziesz musiał uruchomić go kilka razy, zanim będziesz miał szczęście i zobaczysz zwycięską wiadomość w wynikach!

```
You win!
```

Odpowiedz, gdy szczęśliwa liczba nie została wyrzucona

Brak informacji zwrotnej z programu, jeśli użytkownik nie wygrał, może sprawić, że będą się zastanawiać, czy program jest uszkodzony. Dobrą praktyką jest zawsze udzielanie odpowiedzi, gdy użytkownik coś zrobi. W przypadku programu Lucky Dice Roller możesz poinformować ich, że nie wygrali, używając `else`oświadczenia.

1. Dodaj `else`oświadczenie do wydrukowania `"You didn't win, try again!"`.

```
fun main() {
   val myFirstDice = Dice(6)
   val rollResult = myFirstDice.roll()
   val luckyNumber = 4

   if (rollResult == luckyNumber) {
      println("You win!")
   } else {
      println("You didn't win, try again!")
   }
}
```

1. Uruchom program, a bez względu na wynik, Twoi użytkownicy będą zawsze powiadamiani.

W tym momencie użytkownicy wiedzą, czy wygrali, czy nie, ale nie wiedzą, dlaczego. Zawsze podawaj użytkownikom informacje, aby zrozumieli wyniki swoich działań! Wyobraź sobie, że Twój program był wnioskiem o pożyczkę. „Nie zostałeś zatwierdzony, ponieważ twoja zdolność kredytowa jest niska” jest o wiele bardziej pouczające niż „Przepraszam, nie masz pożyczki, spróbuj ponownie!” W przypadku Lucky Dice Roller możesz przekazać użytkownikom inny komunikat informacyjny dla każdego rzutu, jeśli przegrali. Aby to osiągnąć, użyj wielu `else if`instrukcji.

1. Dodaj `else if`instrukcje, aby wydrukować inną wiadomość dla każdej rolki. W razie potrzeby odnieś się do formatu, którego nauczyłeś się w poprzednim zadaniu.

```
fun main() {
    val myFirstDice = Dice(6)
    val rollResult = myFirstDice.roll()
    val luckyNumber = 4

    if (rollResult == luckyNumber) {
        println("You win!")
    } else if (rollResult == 1) {
        println("So sorry! You rolled a 1. Try again!")
    } else if (rollResult == 2) {
        println("Sadly, you rolled a 2. Try again!")
    } else if (rollResult == 3) {
        println("Unfortunately, you rolled a 3. Try again!")
    } else if (rollResult == 5) {
        println("Don't cry! You rolled a 5. Try again!")
    } else {
        println("Apologies! You rolled a 6. Try again!")
    }
}
```

W powyższym kodzie, ty

- Sprawdź, czy `rollResult`jest to `luckyNumber`.
- Jeśli `rollResult`jest `luckyNumber`, wydrukuj zwycięską wiadomość.
- W przeciwnym razie sprawdź, czy `rollResult`jest 1, a jeśli tak, wydrukuj komunikat Spróbuj ponownie.
- W przeciwnym razie sprawdź, czy `rollResult`jest 2, a jeśli tak, wydrukuj inny komunikat Spróbuj ponownie.
- W przeciwnym razie sprawdzaj numer 5.
- Jeśli liczba nie jest żadną z 1–5, jedyną pozostałą opcją jest 6, więc nie ma potrzeby wykonywania kolejnego testu z `else if`, a ostatnią opcję można po prostu złapać za pomocą `else`instrukcji końcowej.

Porada: Możesz mieć tylko jedną `if`instrukcję z jedną `else`instrukcją w bloku kodu if-else, ale pomiędzy nimi możesz mieć tyle `else if`instrukcji, ile potrzebujesz.

Ponieważ posiadanie wielu `else if`przypadków jest bardzo powszechne, Kotlin ma prostszy sposób ich pisania.

4. Użyj wyrażenia „gdy”

Testowanie wielu różnych wyników lub przypadków jest bardzo powszechne w programowaniu. Czasami lista możliwych wyników może być bardzo długa. Na przykład, jeśli rzucasz kostką 12-ścienną, `else if`między sukcesem a finałem będzie 11 stwierdzeń `else`. Aby ułatwić pisanie i czytanie tego rodzaju oświadczeń, co pomaga uniknąć błędów, Kotlin udostępnia `when`oświadczenie.

Zamierzasz zmienić swój program tak, aby używał `when`instrukcji. `when`Instrukcja zaczyna się od słowa kluczowego , `when`po którym następuje nawias `()`. Wewnątrz nawiasów znajduje się wartość do przetestowania. Po nim następuje nawias klamrowy `{}`, aby kod był wykonywany w różnych warunkach.

1. W swoim programie w programie `main()`wybierz kod od pierwszej `if`instrukcji do nawiasu klamrowego `}`, który zamyka ostatnią `else`instrukcję, i usuń go.

```
fun main() {
   val myFirstDice = Dice(6)
   val rollResult = myFirstDice.roll()
   val luckyNumber = 4
}
```

1. W `main()`, poniżej deklaracji `luckyNumber`,
   utwórz `when`oświadczenie. Ponieważ `when`musisz przeprowadzić test z wynikiem rzutu, umieść `rollResult`w nawiasach `()`. Dodaj nawiasy klamrowe `{}`z dodatkowymi odstępami, jak pokazano poniżej.

```
fun main() {
   val myFirstDice = Dice(6)
   val rollResult = myFirstDice.roll()
   val luckyNumber = 4

   when (rollResult) {

   }
}
```

Tak jak poprzednio, najpierw sprawdź, czy `rollResult`jest to samo co `luckyNumber`.

1. Wewnątrz nawiasów klamrowych `{}`tej `when`instrukcji dodaj stwierdzenie, które testuje `rollResult`z `luckyNumber`, a jeśli są takie same, wydrukuj zwycięską wiadomość. Oświadczenie wygląda tak:

```
luckyNumber -> println("You win!")
```

To znaczy:

- Najpierw podajesz wartość, do której porównujesz `rollResult`. To jest `luckyNumber`.

- Postępuj zgodnie ze strzałką (`->`).
- Następnie dodaj akcję do wykonania, jeśli istnieje dopasowanie.

Przeczytaj to jako „Jeśli `rollResult`jest `luckyNumber`, wydrukuj `"You win!"`wiadomość”.

Twój `main()`kod wygląda tak.

```
fun main() {
  val myFirstDice = Dice(6)
  val rollResult = myFirstDice.roll()
  val luckyNumber = 4

  when (rollResult) {
    luckyNumber -> println("You win!")
  }
}
```

1. Użyj tego samego wzoru, aby dodać wiersze i komunikaty dla możliwych rzutów 1–6 z wyjątkiem 4, jak pokazano poniżej. Twoja ukończona `main()`funkcja powinna wyglądać tak.

```
fun main() {
  val myFirstDice = Dice(6)
  val rollResult = myFirstDice.roll()
  val luckyNumber = 4

  when (rollResult) {
    luckyNumber -> println("You won!")
    1 -> println("So sorry! You rolled a 1. Try again!")
    2 -> println("Sadly, you rolled a 2. Try again!")
    3 -> println("Unfortunately, you rolled a 3. Try again!")
    5 -> println("Don't cry! You rolled a 5. Try again!")
    6 -> println("Apologies! You rolled a 6. Try again!")
  }
}
```

1. Uruchom swój program. Nie ma różnicy w wynikach, ale Twój kod jest znacznie bardziej zwarty i łatwiejszy do odczytania.

Gratulacje! Nauczyłeś się dwóch sposobów drukowania wiadomości w zależności od stanu. To potężne narzędzie do pisania ciekawych programów!

5. Kod rozwiązania

```
fun main() {
  val myFirstDice = Dice(6)
  val rollResult = myFirstDice.roll()
```

```
  val luckyNumber = 4

  when (rollResult) {
    luckyNumber -> println("You won!")
    1 -> println("So sorry! You rolled a 1. Try again!")
    2 -> println("Sadly, you rolled a 2. Try again!")
    3 -> println("Unfortunately, you rolled a 3. Try again!")
    5 -> println("Don't cry! You rolled a 5. Try again!")
    6 -> println("Apologies! You rolled a 6. Try again!")
  }
}

class Dice(val numSides: Int) {
  fun roll(): Int {
    return (1..numSides).random()
  }
}
```

6. Podsumowanie

- Użyj `if`instrukcji, aby ustawić warunek wykonania niektórych instrukcji. Na przykład, jeśli użytkownik wyrzuci szczęśliwą liczbę, wydrukuj zwycięską wiadomość.
- Typ `Boolean`danych ma wartości `true`i `false`może być używany do podejmowania decyzji.
- Porównaj wartości za pomocą operatorów, takich jak większa niż (`>`), mniejsza niż (`<`) i równa (`==`).
- Użyj łańcucha `else if`instrukcji, aby ustawić wiele warunków. Na przykład, wydrukuj inną wiadomość dla każdego możliwego rzutu kostką.
- Użyj `else`oświadczenia na końcu łańcucha warunków, aby wyłapać przypadki, które mogą nie być wyraźnie omówione. Jeśli pokryjesz przypadki dla kostek 6-ściennych, `else`stwierdzenie złapie liczby 7 i 8 wyrzucone kostką 8-ścienną.
- Użyj `when`instrukcji jako zwięzłej formy wykonywania kodu na podstawie porównania wartości.

Ogólna forma if-else:

```
if (condition-is-true) {
  execute-this-code
} else if (condition-is-true) {
  execute-this-code
} else {
  execute-this-code
}
```

Kiedy oświadczenie:

```
when (variable) {
   matches-value -> execute-this-code
   matches-value -> execute-this-code
   ...
}
```

7. Dowiedz się więcej

- Słownictwo dla Androida Podstawy w Kotlin
- Kotlin: Kontrola przepływu
- Kotlin: kiedy
- Warunkowe

8. Ćwicz na własną rękę

Uwaga: Praktyki są opcjonalne. Dają ci możliwość przećwiczenia tego, czego nauczyłeś się podczas tego ćwiczenia z programowania, a czasami zawierają wyzwanie kodowania.

Wykonaj następujące czynności:

1. Zmień `myFirstDice`na 8 stron i uruchom swój kod. Co się dzieje?

Podpowiedź: Po zwiększeniu liczby stron `when`wyciąg nie obejmuje już wszystkich przypadków, więc w przypadku niepokrytych przypadków nic nie jest drukowane.

1. Napraw `when`zestawienie, aby uwzględnić wszystkie 8 stron. Możesz to zrobić, dodając przypadki dla dodatkowych numerów. **Wyzwanie:** Zamiast dodawać nowy przypadek dla każdej liczby, użyj `else`instrukcji, aby wychwycić wszystkie przypadki, które nie są wyraźnie omówione.

Wskazówka: możesz dodać więcej skrzynek, aby pokryć więcej boków. Jest to dobry sposób, aby to zrobić, jeśli chcesz mieć inny komunikat dla każdej liczby, którą można wyrzucić. Możesz też użyć `else`instrukcji i wydrukować tę samą wiadomość dla wszystkich stron większych niż 6 objętych bieżącym kodem.

1. Zmień `myFirstDice`, aby mieć tylko 4 strony. Co się dzieje?

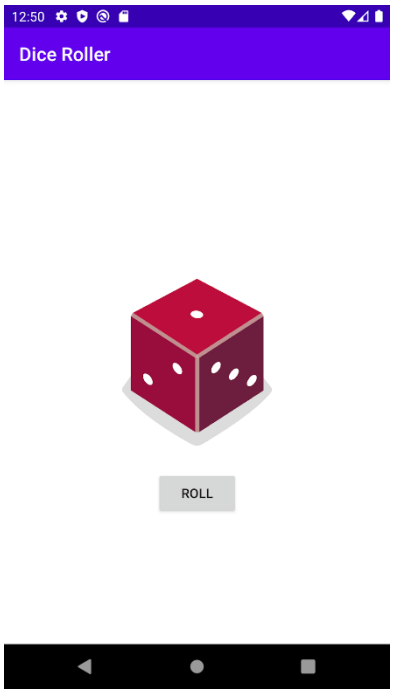
Podpowiedź: Zmiana liczby boków kostek na mniej niż to, co obejmuje `when`stwierdzenie, nie ma zauważalnego efektu, ponieważ wszystkie przypadki, które mogą wystąpić, są objęte.

Dodaj obrazy do aplikacji Dice Roller

1. Zanim zaczniesz

W tym ćwiczeniu z kodowania dodasz obrazy kości do istniejącej aplikacji Dice Roller na Androida. Pamiętaj, aby najpierw ukończyć wcześniejsze ćwiczenia z programowania dotyczące tworzenia podstaw aplikacji Dice Roller.

Zamiast wyświetlać wartość rzutu kostką w `TextView`, Twoja aplikacja wyświetli obraz kości odpowiedni dla liczby boków, które zostały wyrzucone. Będzie to znacznie bardziej wizualne i ulepszone wrażenia użytkownika dla Twojej aplikacji.



Otrzymasz link do pobrania obrazów kostek i dodasz je jako zasoby w swojej aplikacji. Aby napisać kod, którego obrazu kości użyć, użyjesz `when`instrukcji w Kotlinie.

Warunki wstępne

- Ukończono ćwiczenia z kodowania w aplikacji Stwórz interaktywną grę w kości.
- Potrafi pisać instrukcje przepływu sterowania (`if / else`, `when`instrukcje).
- Możliwość aktualizacji interfejsu użytkownika aplikacji na podstawie danych wprowadzonych przez użytkownika (modyfikacja `MainActivity.kt`pliku).
- Możliwość dodania odbiornika kliknięć do`Button`.
- Możliwość dodawania zasobów graficznych do aplikacji na Androida.

Czego się nauczysz

- Jak zaktualizować, `ImageView`gdy aplikacja jest uruchomiona.

- Jak dostosować zachowanie aplikacji na podstawie różnych warunków (za pomocą `when`instrukcji).

Co zbudujesz

- Dice Roller Android aplikacja, która ma `Button`rzucać kostką i aktualizować obraz na ekranie.

Czego potrzebujesz

- Komputer z zainstalowanym Android Studio.
- Połączenie internetowe w celu pobrania obrazów kostek.

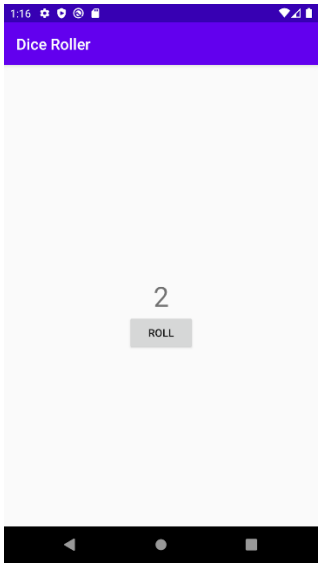
2. Zaktualizuj układ aplikacji

W tym zadaniu zastąpisz `TextView`w swoim układzie symbolem, `ImageView`który wyświetla obraz wyniku rzutu kostką.

Otwórz aplikację Dice Roller

1. Otwórz i uruchom aplikację Dice Roller z poprzedniego ćwiczenia kodowania w Android Studio. Możesz użyć kodu rozwiązania lub utworzonego kodu.

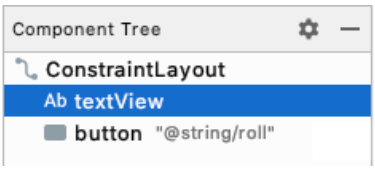
Aplikacja powinna wyglądać tak.



1. Otwórz `activity_main.xml`(**app > res > layout > activity_main.xml**). Spowoduje to otwarcie **Edytora układu** .

Usuń widok tekstu

1. W **Edytorze układu** wybierz `TextView`w **Drzewie komponentów** .

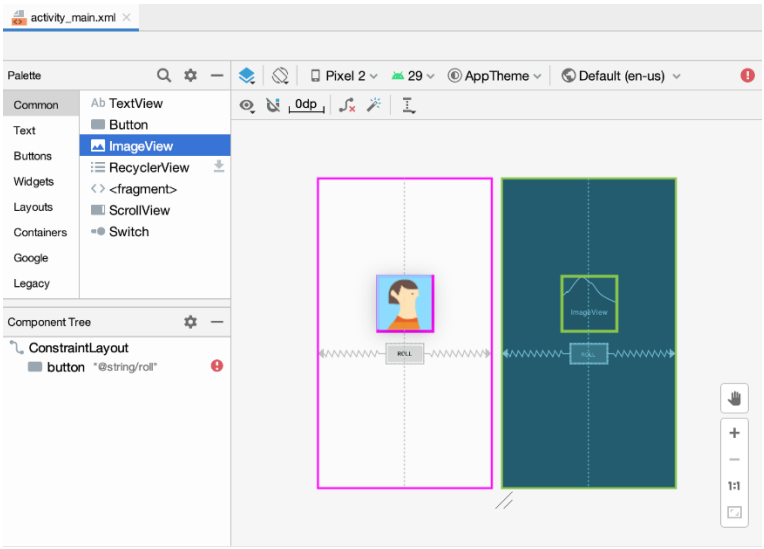


Wskazówka: w miarę dodawania kolejnych komponentów interfejsu użytkownika oraz dodawania i usuwania wiązań, możesz tymczasowo znaleźć jeden z nich `View`, który nakłada się na drugi, co utrudnia wybranie tego z tyłu. W takim przypadku możesz wybrać a `View`, wybierając go w **drzewie komponentów** .

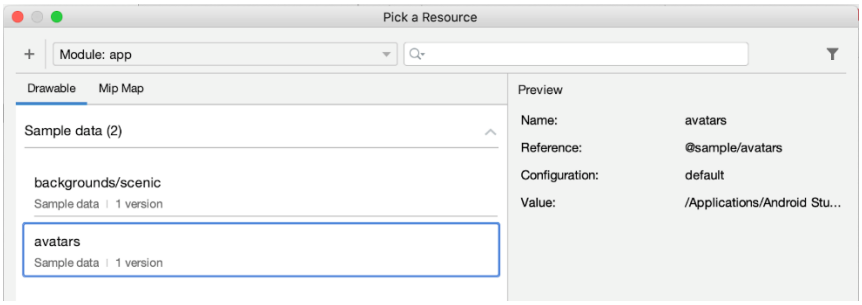
1. Kliknij prawym przyciskiem myszy i wybierz **Usuń** lub naciśnij `Delete`klawisz.
2. Na razie zignoruj ostrzeżenie `Button`. Naprawisz to w następnym kroku.

Dodaj ImageView do układu

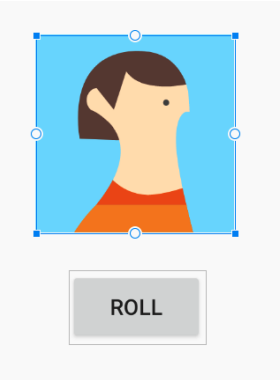
1. Przeciągnij ikonę `ImageView`z **Palety** do widoku **Projekt** , umieszczając ją nad `Button`.



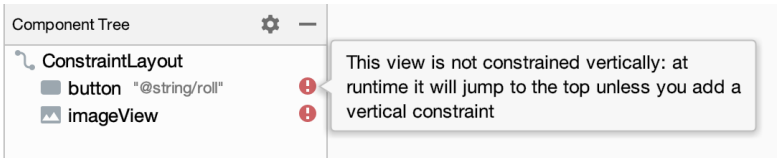
1. W oknie dialogowym **Wybierz zasób** wybierz **awatary** w obszarze **Dane przykładowe** . To jest tymczasowy obraz, którego będziesz używać, dopóki nie dodasz obrazów kostek w następnym zadaniu.



1. Naciśnij **OK** . Widok **projektu** Twojej aplikacji powinien wyglądać tak.



1. W **drzewie komponentów** zauważysz dwa błędy. Nie `Button`jest ograniczony w pionie i `ImageView`nie jest ani w pionie, ani w poziomie.

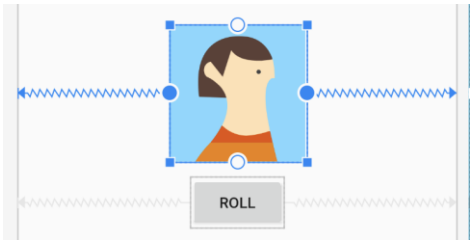


Nie `Button`jest ograniczony w pionie, ponieważ usunięto element, `TextView`poniżej którego był pierwotnie umieszczony. Teraz musisz ustawić `ImageView`i `Button`poniżej.

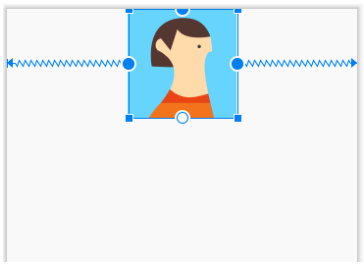
Ustaw ImageView i przycisk

Musisz wyśrodkować `ImageView`w pionie obraz na ekranie, niezależnie od tego, gdzie się `Button`znajduje.

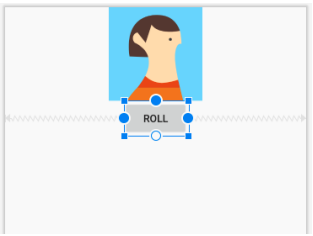
1. Dodaj wiązania poziome do `ImageView`. Połącz lewą stronę z `ImageView`lewą krawędzią rodzica `ConstraintLayout`.
2. Połącz prawą stronę z `ImageView`prawą krawędzią rodzica. Spowoduje to wyśrodkowanie `ImageView`w poziomie rodzica.



1. Dodaj wiązanie pionowe do `ImageView`, łączące górną część z `ImageView`górną częścią rodzica. Przesunie `ImageView`się do góry `ConstraintLayout`.



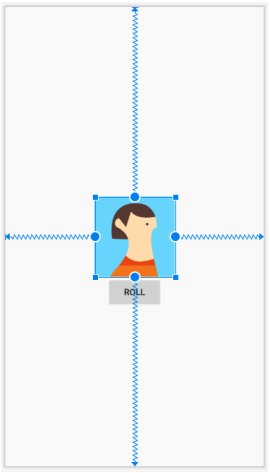
1. Dodaj wiązanie pionowe do `Button`, łączące górną część `Button`z dolną częścią `ImageView`. Będzie przesuwać się `Button`pod `ImageView`.



1. Teraz wybierz `ImageView`ponownie i dodaj wiązanie pionowe łączące dolną `ImageView`część elementu macierzystego z dołem elementu macierzystego. To wyśrodkowuje `ImageView`pionowo w `ConstraintLayout`.

Wszystkie ostrzeżenia o ograniczeniach powinny teraz zniknąć.

Po tym wszystkim widok **Projekt** powinien wyglądać tak, z `ImageView`tym pośrodku i `Button`tuż pod nim.



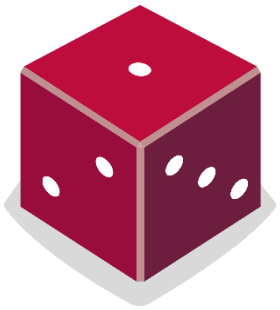
Możesz zauważyć ostrzeżenie `ImageView`w **drzewie komponentów** , które mówi, aby dodać opis zawartości do twojego `ImageView`. Na razie nie przejmuj się tym ostrzeżeniem, ponieważ w dalszej części ćwiczenia z kodowania będziesz ustawiać opis zawartości na `ImageView`podstawie wyświetlanego obrazu kości. Ta zmiana zostanie wprowadzona w kodzie Kotlin.

3. Dodaj obrazy kości

W tym zadaniu pobierzesz kilka obrazów kostek i dodasz je do swojej aplikacji.

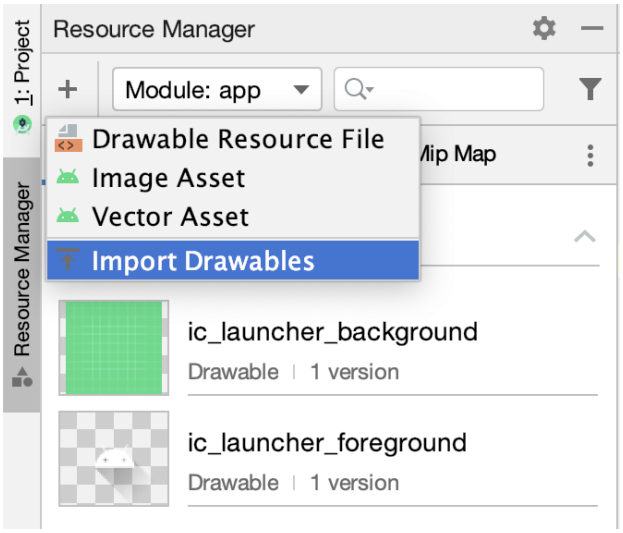
Pobierz obrazy kości

1. Otwórz ten adres URL , aby pobrać plik zip z obrazami kostek na swój komputer. Poczekaj na zakończenie pobierania.
2. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
3. Kliknij dwukrotnie plik zip, aby go rozpakować. Spowoduje to utworzenie nowego `dice_images`folderu zawierającego 6 plików obrazów kości, wyświetlających wartości kości od 1 do 6.

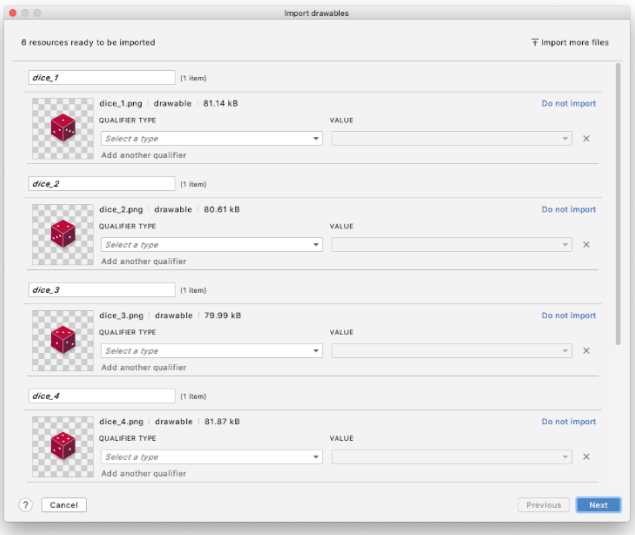


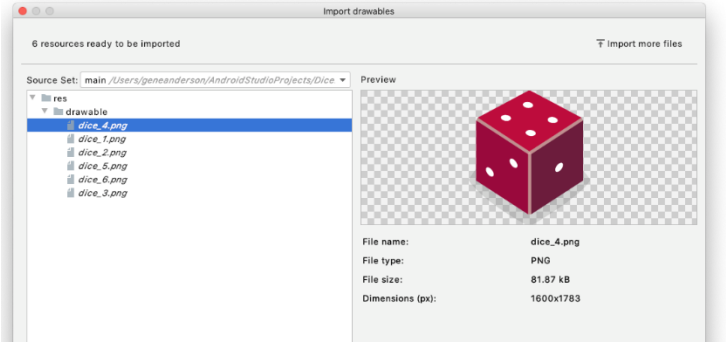
Dodaj obrazy kości do swojej aplikacji

1. W Android Studio kliknij w menu **Widok > Narzędzia Windows > Menedżer zasobów lub kliknij kartę Menedżer zasobów** po lewej stronie okna **projektu** .
2. Kliknij przycisk **+** poniżej **Menedżera zasobów** i wybierz opcję **Importuj elementy do rysowania** . Spowoduje to otwarcie przeglądarki plików.

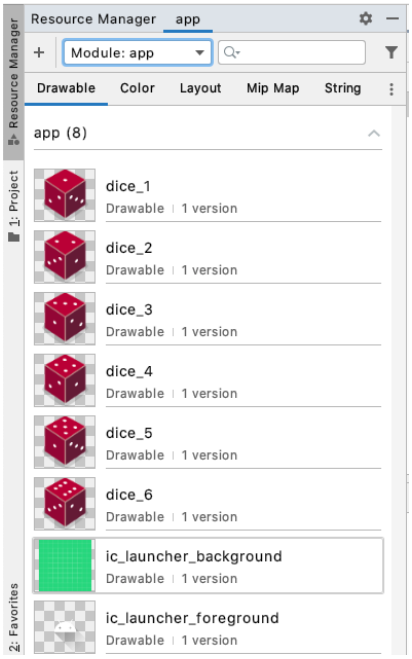


1. Znajdź i wybierz pliki obrazów 6 kostek. Możesz wybrać pierwszy plik, a następnie trzymając wciśnięty Shiftklawisz wybrać pozostałe pliki.
2. Kliknij **Otwórz** .
3. Kliknij **Dalej** , a następnie **Importuj** , aby potwierdzić, że chcesz zaimportować te 6 zasobów.





1. Jeśli pliki zostały pomyślnie zaimportowane, w Menedżerze zasobów (**app>res>drawable)** dla Twojej aplikacji powinno pojawić się 6 obrazów.



Dobra robota! W następnym zadaniu wykorzystasz te obrazy w swojej aplikacji.

Ważny! - Będziesz mógł odwoływać się do tych obrazów w swoim kodzie Kotlin za pomocą ich identyfikatorów zasobów:

- `R.drawable.dice_1`
- `R.drawable.dice_2`
- `R.drawable.dice_3`
- `R.drawable.dice_4`
- `R.drawable.dice_5`
- `R.drawable.dice_6`

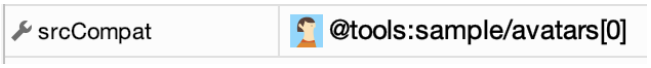
4. Użyj obrazów kości

Zastąp przykładowy obraz awatara

1. W **Edytorze projektu** wybierz `ImageView`.
2. W sekcji **Atrybuty** w sekcji **Zadeklarowane atrybuty znajdź atrybut srcCompat** narzędzia , który jest ustawiony na obraz awatara.

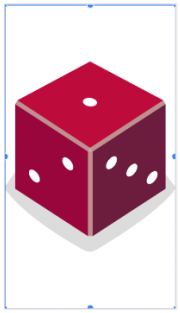
Pamiętaj, że atrybut tools **srcCompat** używa dostarczonego obrazu tylko w widoku **Projekt** w Android Studio. Obraz jest wyświetlany tylko programistom podczas tworzenia aplikacji, ale nie będzie widoczny, gdy faktycznie uruchomisz aplikację w emulatorze lub na urządzeniu.

1. Kliknij mały podgląd awatara. Spowoduje to otwarcie okna dialogowego, w którym można wybrać nowy zasób do wykorzystania w tym celu `ImageView`.



1. Wybierz `dice_1`rysunek i kliknij **OK** .

Łał! `ImageView`Zajmuje cały ekran .



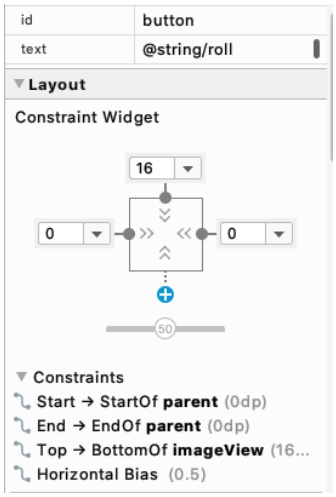
Następnie dostosujesz szerokość i wysokość `ImageView`, aby nie ukrywać `Button`.

1. W oknie **Atrybuty** w obszarze **Widget Ograniczenia** zlokalizuj atrybuty **layout_width** i **layout_height** . Obecnie są ustawione na **wrap_content** , co oznacza, że będą tak wysokie i tak szerokie, jak zawartość (obraz źródłowy) w środku.`ImageView`
2. Zamiast tego ustaw stałą szerokość 160dp i stałą wysokość 200dp w `ImageView`. Naciśnij **Enter** .

Teraz jest `ImageView` znacznie mniejszy.

Może się okazać, że `Button` jest trochę za blisko obrazu.

1. Dodaj górny margines do przycisku 16dp, ustawiając go w Widżecie **Ograniczenia** .



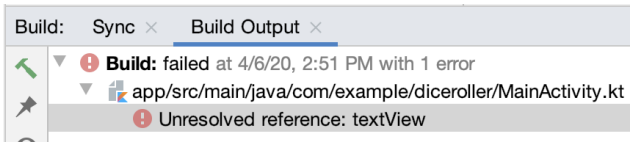
Po zaktualizowaniu widoku **Projekt aplikacja wygląda znacznie lepiej!**

Uwaga: Użyj pikseli niezależnych od gęstości (dp) jako jednostki, aby zdefiniować te wymiary, aby rozmiar obrazu był odpowiednio skalowany na urządzeniach o różnej rozdzielczości pikseli.

Zmień obraz kości po kliknięciu przycisku

Układ został naprawiony, ale `MainActivity`klasa musi zostać zaktualizowana, aby używać obrazów kostek.

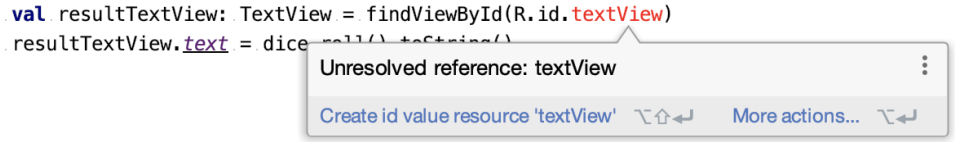
Obecnie w aplikacji w `MainActivity.kt`pliku występuje błąd. Jeśli spróbujesz uruchomić aplikację, zobaczysz ten błąd kompilacji:



Dzieje się tak, ponieważ Twój kod nadal odwołuje się do `TextView`tego, który usunąłeś z układu.

1. Otwórz `MainActivity.kt`(**app > java > com.example.diceroller > MainActivity.kt**)

Kod odnosi się do `R.id.textView`, ale Android Studio go nie rozpoznaje.



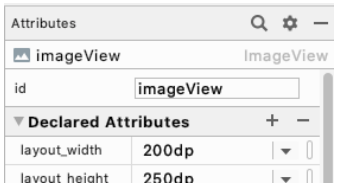
1. W ramach `rollDice()`metody wybierz dowolny kod, który się do niego odwołuje `TextView`i usuń go.

```
// Update the TextView with the dice roll
val resultTextView: TextView = findViewById(R.id.textView)
resultTextView.text = dice.roll().toString()
```

1. Nadal w obrębie `rollDice()`, utwórz nową zmienną o nazwie `diceImage`type `ImageView`. Ustaw go na równy `ImageView`z układu. Użyj `findViewById()`metody i przekaż identyfikator zasobu dla `ImageView`, `R.id.imageView`jako argument wejściowy.

```
val diceImage: ImageView = findViewById(R.id.imageView)
```

Jeśli zastanawiasz się, jak ustalić dokładny identyfikator zasobu `ImageView`, sprawdź **identyfikator** u góry okna **Atrybuty** .



Kiedy odwołujesz się do tego identyfikatora zasobu w kodzie Kotlin, upewnij się, że wpisujesz go dokładnie tak samo (małe i, duże V, bez spacji). W przeciwnym razie Android Studio wyświetli błąd.

1. Dodaj ten wiersz kodu, aby sprawdzić, czy można poprawnie zaktualizować `ImageView`po kliknięciu przycisku. Rzut kostką nie zawsze będzie miał wartość „2”, ale po prostu użyj `dice_2`obrazu do celów testowych.

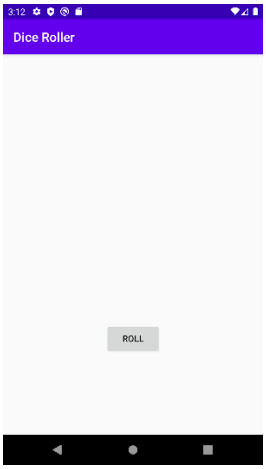
```
diceImage.setImageResource(R.drawable.dice_2)
```

Ten kod wywołuje `setImageResource()`metodę na `ImageView`, przekazując identyfikator zasobu dla `dice_2`obrazu. Spowoduje to zaktualizowanie `ImageView`na ekranie wyświetlania `dice_2`obrazu.

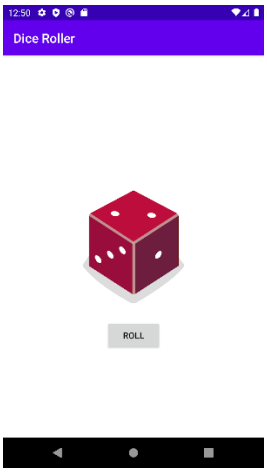
Metoda rollDice() powinna teraz wyglądać tak:

```
private fun rollDice() {
   val dice = Dice(6)
   val diceRoll = dice.roll()
   val diceImage: ImageView = findViewById(R.id.imageView)
   diceImage.setImageResource(R.drawable.dice_2)
}
```

1. Uruchom aplikację, aby sprawdzić, czy działa bez błędów. Aplikacja powinna zaczynać się od pustego ekranu z wyjątkiem przycisku **Roll** .



Po naciśnięciu przycisku pojawi się obraz kości z wartością 2. Tak!!



Byłeś w stanie zmienić obraz na podstawie dotknięcia przycisku! Zbliżasz się!

5. Wyświetl prawidłowy obraz kości na podstawie rzutu kostką

Oczywiście wynik rzutu kośćmi nie zawsze będzie równy 2. Użyj logiki przepływu sterowania, której nauczyłeś się podczas ćwiczeń z kodowania Add Conditional Behavior for Different Dice Rolls , aby odpowiedni obraz kości był wyświetlany na ekranie w zależności od losowego rzutu kośćmi.

Zanim zaczniesz pisać kod, zastanów się koncepcyjnie nad tym, jak aplikacja powinna się zachowywać, pisząc *pseudokod* opisujący, co powinno się wydarzyć. Na przykład:

Jeśli użytkownik wyrzuci 1, wyświetl `dice_1`obraz.

Jeśli użytkownik wyrzuci 2, wyświetl `dice_2`obraz.

itp...

Uwaga: *Pseudokod* to nieformalny opis tego, jak może działać jakiś kod. Wykorzystuje niektóre elementy języka komputerowego, takie jak `if / else`, ale opisuje rzeczy w sposób zrozumiały dla człowieka. Może to być przydatne do zaplanowania właściwego podejścia, które należy podjąć, zanim wszystkie szczegóły zostaną ustalone.

Powyższy pseudokod można napisać za pomocą `if / else`instrukcji w Kotlinie na podstawie wartości rzutu kostką.

```
if (diceRoll == 1) {
  diceImage.setImageResource(R.drawable.dice_1)
} else if (diceRoll == 2) {
  diceImage.setImageResource(R.drawable.dice_2)
}
 ...
```

`if / else`Jednak pisanie dla każdego przypadku jest dość powtarzalne. Tę samą logikę można w prostszy sposób wyrazić za pomocą `when`stwierdzenia. To jest bardziej zwięzłe (mniej kodu)! Użyj tego podejścia w swojej aplikacji.

```
when (diceRoll) {
  1 -> diceImage.setImageResource(R.drawable.dice_1)
  2 -> diceImage.setImageResource(R.drawable.dice_2)
  ...
```

Zaktualizuj metodę rollDice( )

1. W tej `rollDice()`metodzie usuń wiersz kodu, który za każdym razem ustawia identyfikator zasobu obrazu na `dice_2`obraz.

```
diceImage.setImageResource(R.drawable.dice_2)
```

1. Zastąp go `when`instrukcją, która aktualizuje wartość `ImageView`na podstawie `diceRoll`wartości.

```
  when (diceRoll) {
    1 -> diceImage.setImageResource(R.drawable.dice_1)
    2 -> diceImage.setImageResource(R.drawable.dice_2)
    3 -> diceImage.setImageResource(R.drawable.dice_3)
    4 -> diceImage.setImageResource(R.drawable.dice_4)
    5 -> diceImage.setImageResource(R.drawable.dice_5)
    6 -> diceImage.setImageResource(R.drawable.dice_6)
  }
```

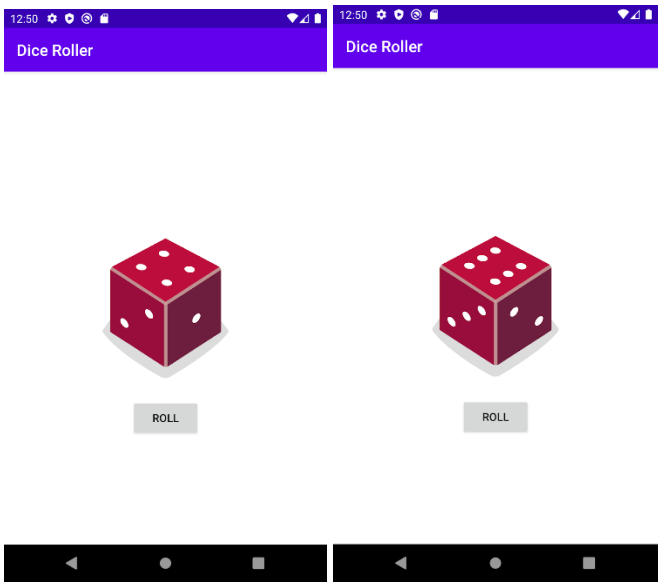
Metoda `rollDice()`powinna wyglądać tak, gdy skończysz ze zmianami.

```
private fun rollDice() {
  val dice = Dice(6)
  val diceRoll = dice.roll()

  val diceImage: ImageView = findViewById(R.id.imageView)

  when (diceRoll) {
    1 -> diceImage.setImageResource(R.drawable.dice_1)
    2 -> diceImage.setImageResource(R.drawable.dice_2)
    3 -> diceImage.setImageResource(R.drawable.dice_3)
    4 -> diceImage.setImageResource(R.drawable.dice_4)
    5 -> diceImage.setImageResource(R.drawable.dice_5)
    6 -> diceImage.setImageResource(R.drawable.dice_6)
  }
}
```

1. Uruchom aplikację. Kliknięcie przycisku **Rzuć** zmienia obraz kości na inne wartości poza 2. To działa!



Zoptymalizuj swój kod

Jeśli chcesz napisać jeszcze bardziej zwięzły kod, możesz wprowadzić następującą zmianę kodu. Nie ma to żadnego widocznego wpływu na użytkownika Twojej aplikacji, ale sprawi, że Twój kod będzie krótszy i mniej powtarzalny.

Być może zauważyłeś, że wezwanie do `diceImage.setImageResource()`pojawia się 6 razy w Twoim oświadczeniu when.

```
when (diceRoll) {
   1 -> diceImage.setImageResource(R.drawable.dice_1)
   2 -> diceImage.setImageResource(R.drawable.dice_2)
   3 -> diceImage.setImageResource(R.drawable.dice_3)
   4 -> diceImage.setImageResource(R.drawable.dice_4)
   5 -> diceImage.setImageResource(R.drawable.dice_5)
   6 -> diceImage.setImageResource(R.drawable.dice_6)
}
```

Jedyną rzeczą, która zmienia się między każdym przypadkiem, jest używany identyfikator zasobu. Oznacza to, że możesz utworzyć zmienną do przechowywania używanego identyfikatora zasobu. Następnie możesz wywołać `diceImage.setImageResource()`tylko raz w swoim kodzie i przekazać poprawny identyfikator zasobu.

1. Zastąp powyższy kod następującym.

```
val drawableResource = when (diceRoll) {
  1 -> R.drawable.dice_1
  2 -> R.drawable.dice_2
  3 -> R.drawable.dice_3
  4 -> R.drawable.dice_4
  5 -> R.drawable.dice_5
  6 -> R.drawable.dice_6
}

diceImage.setImageResource(drawableResource)
```

Nowa koncepcja polega na tym, że `when`wyrażenie może faktycznie zwrócić wartość. W przypadku tego nowego fragmentu kodu `when`wyrażenie zwraca poprawny identyfikator zasobu, który zostanie zapisany w `drawableResource`zmiennej. Następnie możesz użyć tej zmiennej do aktualizacji wyświetlanego zasobu obrazu.

1. Zauważ, że `when`jest teraz podkreślone na czerwono. Jeśli najedziesz na niego wskaźnikiem, zobaczysz komunikat o błędzie: **wyrażenie „kiedy” musi być wyczerpujące, dodaj niezbędną gałąź „else”** .

Błąd polega na tym, że wartość `when`wyrażenia jest przypisana do `drawableResource`, więc `when`musi być wyczerpująca — musi obsługiwać wszystkie możliwe przypadki, aby zawsze była zwracana wartość, nawet jeśli zmienisz kostkę na 12-ścienne. Android Studio sugeruje dodanie `else`oddziału. Możesz to naprawić, zmieniając wielkość liter `6`na `else`. Sprawy za `1`pośrednictwem `5`są takie same, ale wszystkie inne, w tym `6`są obsługiwane przez `else`.

```
val drawableResource = when (diceRoll) {
  1 -> R.drawable.dice_1
  2 -> R.drawable.dice_2
  3 -> R.drawable.dice_3
  4 -> R.drawable.dice_4
  5 -> R.drawable.dice_5
  else -> R.drawable.dice_6
}

diceImage.setImageResource(drawableResource)
```

1. Uruchom aplikację, aby upewnić się, że nadal działa poprawnie. Przetestuj to wystarczająco, aby upewnić się, że wszystkie liczby pojawiają się na obrazach kostek od 1 do 6.

Ustaw odpowiedni opis zawartości w ImageView

Teraz, gdy zastąpiłeś wylosowaną liczbę obrazem, czytniki ekranu nie mogą już powiedzieć, jaki numer został wyrzucony. Aby rozwiązać ten problem, po zaktualizowaniu zasobu obrazu zaktualizuj opis zawartości `ImageView`. Opis treści powinien być opisem tekstowym tego, co jest wyświetlane w `ImageView`, aby czytniki ekranu mogły to opisać.

```
diceImage.contentDescription = diceRoll.toString()
```

Czytniki ekranu mogą czytać na głos ten opis zawartości, więc jeśli rzut kostką z obrazem „6” zostanie wyświetlony na ekranie, opis zawartości zostanie odczytany na głos jako „6”.

Uwaga: Zwykle opis treści powinien wykorzystywać zasoby tekstowe, które można przetłumaczyć na inne języki, ale zajmiemy się tym w przyszłej lekcji.

6. Zastosuj dobre praktyki kodowania

Stwórz bardziej przydatne środowisko uruchamiania

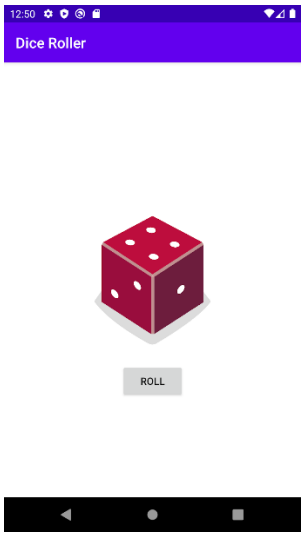
Gdy użytkownik otwiera aplikację po raz pierwszy, aplikacja jest pusta (z wyjątkiem przycisku **Roll**), co wygląda dziwnie. Użytkownicy mogą nie wiedzieć, czego się spodziewać, więc zmień interfejs użytkownika, aby wyświetlał losowy rzut kostką po pierwszym uruchomieniu aplikacji i utworzeniu `Activity`. Wtedy użytkownicy z większym prawdopodobieństwem zrozumieją, że dotknięcie przycisku **Rzuć** spowoduje rzut kostką.

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContentView(R.layout.activity_main)

  val rollButton: Button = findViewById(R.id.button)
  rollButton.setOnClickListener { rollDice() }

  // Do a dice roll when the app starts
```

```
    rollDice()
}
```



Skomentuj swój kod

Dodaj kilka komentarzy do kodu, aby opisać, co dzieje się w napisanym przez Ciebie kodzie.

Po wprowadzeniu wszystkich tych zmian, tak `rollDice()`może wyglądać Twoja metoda.

```
/**
* Roll the dice and update the screen with the result.
*/
private fun rollDice() {
    // Create new Dice object with 6 sides and roll the dice
    val dice = Dice(6)
    val diceRoll = dice.roll()

    // Find the ImageView in the layout
    val diceImage: ImageView = findViewById(R.id.imageView)

    // Determine which drawable resource ID to use based on the dice roll
    val drawableResource = when (diceRoll) {
        1 -> R.drawable.dice_1
        2 -> R.drawable.dice_2
        3 -> R.drawable.dice_3
        4 -> R.drawable.dice_4
        5 -> R.drawable.dice_5
        else -> R.drawable.dice_6
    }

    // Update the ImageView with the correct drawable resource ID
    diceImage.setImageResource(drawableResource)
```

```
    // Update the content description
    diceImage.contentDescription = diceRoll.toString()
}
```

Aby uzyskać pełny `MainActivity.kt`plik, zobacz kod rozwiązania w serwisie GitHub połączonym w następnym kroku.

Dobra robota po ukończeniu aplikacji Dice Roller! Teraz możesz przenieść tę aplikację na kolejną noc gry ze znajomymi!

7. Kod rozwiązania

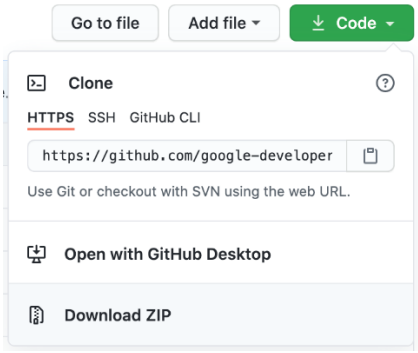
Kod rozwiązania dla tego ćwiczenia programowania znajduje się w projekcie i module pokazanym poniżej.

Adres URL kodu rozwiązania: https://github.com/google-developer-training/android-basics-kotlin-dice-roller-with-images-app-solution

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

Pobierz kod

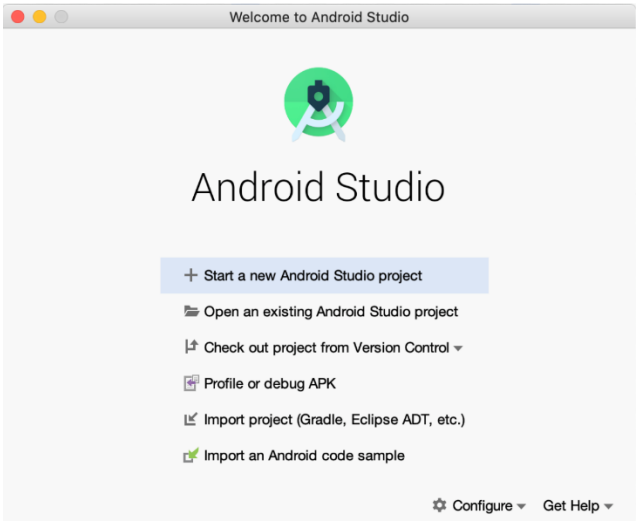
1. Kliknij podany adres URL. Spowoduje to otwarcie strony GitHub projektu w przeglądarce.
2. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli okno dialogowe.



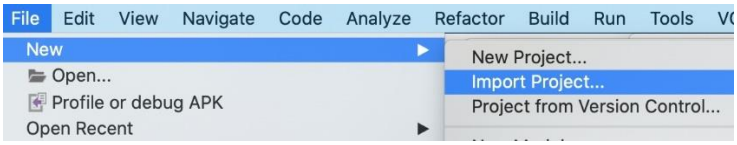
1. W oknie dialogowym kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na swoim komputerze. Poczekaj na zakończenie pobierania.
2. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
3. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego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



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

8. Podsumowanie

- Służy `setImageResource()`do zmiany obrazu wyświetlanego w`ImageView`
- Użyj instrukcji przepływu sterowania, takich jak `if / else`wyrażenia lub `when`wyrażenia, do obsługi różnych przypadków w aplikacji, na przykład wyświetlania różnych obrazów w różnych okolicznościach.

9. Dowiedz się więcej

- Słowniczek Android Kotlin
- `if`wyrażenie w Kotlinie
- `when`wyrażenie w Kotlinie

- `ImageView.setImageResource()`
- Dostępność

10. Ćwicz na własną rękę

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

Wykonaj następujące czynności:

1. Dodaj kolejną kostkę do aplikacji, aby jeden przycisk **Rzut** dawał wyniki 2 kostkami. Ile `ImageViews`będziesz potrzebować w swoim układzie? Jak to wpłynie na `MainActivity.kt`kod?

Sprawdź swoją pracę:

Twoja gotowa aplikacja powinna działać bez błędów i pokazywać dwie kostki.

Napisz testy jednostkowe

1. Zanim zaczniesz

W poprzednich ćwiczeniach z programowania nauczyłeś się tworzyć projekt za pomocą Android Studio, modyfikować XML, aby utworzyć dostosowany interfejs użytkownika dla swojej aplikacji i modyfikować logikę biznesową, aby dodać funkcjonalność. To laboratorium kodowania koncentruje się na tym, dlaczego testowanie jest ważne i obejmuje testy jednostkowe. Masz okazję zobaczyć, jak wyglądają i jak je napisać.

Warunki wstępne

- Utworzyłeś projekt w Android Studio.
- Masz pewne doświadczenie w pisaniu kodu w Android Studio.

Czego się nauczysz

- Dlaczego testowanie jest ważne.
- Jak wyglądają testy jednostkowe.
- Jak pisać i uruchamiać testy jednostkowe.

Co będziesz potrzebował

- Komputer z zainstalowanym Android Studio.

- Projekt utworzony w poprzednim ćwiczeniu z programowania w tej ścieżce.

2. Wstęp

Teraz, gdy już napisałeś już kod na Androida, to świetny czas, aby uzupełnić go o kod testowy. Najpierw zapoznasz się z filozofią testowania, następnie zanurzysz się głębiej w automatycznie generowane testy w projekcie na Androida, a na końcu napiszesz własne testy dla aplikacji Dice Roller! Ta lekcja obejmuje dużo materiału, ale nie daj się zastraszyć! Nie spiesz się z tym materiałem, ponieważ testowanie zajmuje dużo czasu i wymaga dużo praktyki. Nie zniechęcaj się, jeśli nie zrozumiesz tego od razu.

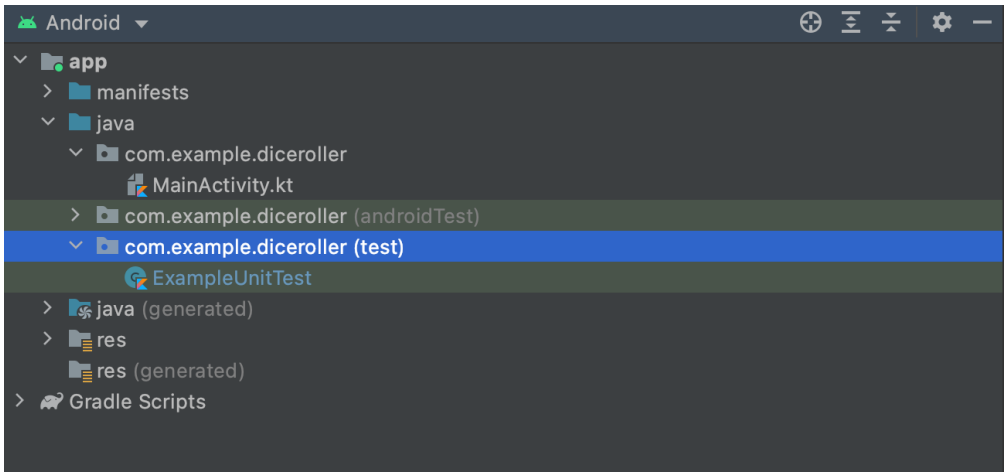
Dlaczego testowanie jest ważne?

Na pierwszy rzut oka może się wydawać, że tak naprawdę nie potrzebujesz testów w swojej aplikacji. Gdy Twoja aplikacja jest mała i ma ograniczoną funkcjonalność, łatwo ją przetestować ręcznie i sprawdzić, czy wszystko działa poprawnie. Jednak wraz z rozwojem aplikacji ręczne testowanie wymaga znacznie więcej wysiłku niż pisanie testów automatycznych. Co więcej, gdy zaczniesz pracować nad aplikacjami na poziomie profesjonalnym, testowanie staje się krytyczne, gdy masz dużą bazę użytkowników. Musisz uwzględnić wiele różnych typów urządzeń z wieloma różnymi wersjami Androida. W końcu dochodzisz do punktu, w którym testy automatyczne mogą uwzględniać większość scenariuszy użytkowania znacznie szybciej niż testy ręczne. Uruchamiając testy przed wydaniem nowego kodu, możesz wprowadzić zmiany w istniejącym kodzie, aby uniknąć wydania aplikacji z nieoczekiwanym zachowaniem. Pamiętaj, że testy automatyczne to testy wykonywane za pomocą oprogramowania, w przeciwieństwie do testów ręcznych, które wykonuje osoba mająca bezpośrednią interakcję z urządzeniem. Testy automatyczne i testy ręczne odgrywają kluczową rolę w zapewnieniu przyjemnego doświadczenia użytkownikom Twojego produktu. Jednak testy automatyczne mogą być bardziej precyzyjne i optymalizują produktywność Twojego zespołu, ponieważ nie wymaga się ich wykonywania od osoby i można je wykonać znacznie szybciej niż test ręczny.

Bliższe spojrzenie na testy jednostkowe

W tym ćwiczeniu z programowania koncentrujesz się na testach jednostkowych. Później omówisz testy oprzyrządowania. Na początek przyjrzyj się testom generowanym podczas tworzenia aplikacji na Androida za pomocą Android Studio. Zdobędziesz także praktyczne doświadczenie w prowadzeniu testów i zapoznasz się z pisaniem kodu testowego.

W poprzedniej ścieżce sprawdziłeś, gdzie znaleźć pliki źródłowe do testów. Testy jednostkowe zawsze znajdują się w `test`katalogu:



1. Otwórz `app/build.gradle` plik i spójrz na zależności. Zobaczysz niektóre zależności oznaczone jako `testImplementation` i `androidTestImplementation`, które odpowiadają odpowiednio testom jednostkowym i oprzyrządowaniu. Na uwagę zasługuje:

app/build.gradle

```
testImplementation 'junit:junit:4.12'
```

Biblioteka `JUnit`, która steruje testami jednostkowymi i pozwala oznaczyć kod jako test, aby można go było skompilować i uruchomić w taki sposób, aby mógł testować kod aplikacji.

1. W `test` katalogu otwórz `ExampleUnitTest.kt` plik.

Powinieneś zobaczyć przykładowy test jednostkowy, który wygląda tak:

ExampleUnitTest.kt

```
class ExampleUnitTest {
  @Test
  fun addition_isCorrect() {
    assertEquals(4, 2 + 2)
  }
}
```

Podczas dodawania kodu do aplikacji Dice Roller prawdopodobnie nie napisałeś żadnych testów. W związku z tym nie ma nic poza ogólnym kodem tworzonym automatycznie przez Android Studio. Jest to arbitralny test, który służy jako miejsce na bardziej odpowiednie testy, które ma napisać programista. Obecnie ten blok kodu testuje tylko, że 2 + 2 = 4. Oczywiście to zawsze prawda. Przyjrzyj się bliżej temu, co się dzieje:

- Funkcje testowe muszą być najpierw opatrzone `@Test` adnotacją zaimportowaną z `org.junit.test` biblioteki. Możesz myśleć o adnotacji jako o tagach metadanych dla fragmentu kodu, który może zmienić sposób kompilowania kodu. W takim przypadku `@Test` adnotacja informuje kompilator, że poniższa metoda jest testem, co umożliwia jej uruchomienie jako takie.

Po adnotacji masz deklarację funkcji, w tym przypadku `addition_isCorrect()` funkcji. Wewnątrz funkcji `assertEquals()` funkcja zapewnia, że oczekiwana wartość powinna być równa rzeczywistej

wartości uzyskanej za pomocą logiki biznesowej. Ostatecznym celem testu jednostkowego są metody asercji. Ostatecznie chcesz zapewnić, że wynik uzyskany z twojego kodu jest w określonym stanie. Jeśli stan wyniku jest zgodny ze stanem oczekiwanym, test kończy się pomyślnie. Jeśli stan wyniku nie jest zgodny z oczekiwanym stanem, test kończy się niepowodzeniem. W tym przypadku kod porównuje dwie wartości, więc`assertEquals()`Metoda przyjmuje dwa parametry — wartość oczekiwaną i wartość rzeczywistą. Zgodnie ze swoją nazwą oczekiwana wartość jest tym, czego oczekujesz od konkretnego wyniku, w tym przypadku 4. Rzeczywista wartość reprezentuje wynik rzeczywistego fragmentu kodu. Ogólnie rzecz biorąc, testuje to fragment kodu z samej aplikacji. W tym przypadku jest to tylko dowolny fragment kodu, na przykład `2 + 2`. Bez dalszych ceregieli uruchom ten test, aby zobaczyć, co się stanie.

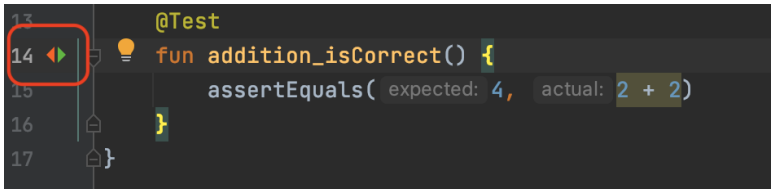
Uwaga : W `JUnit`bibliotece znajduje się wiele asercji. Niektóre typowe twierdzenia, z którymi możesz się spotkać, to:

- `assertEquals()`
- `assertNotEquals()`
- `assertThat()`
- `assertTrue()`
- `assertFalse()`
- `assertNull()`
- `assertNotNull()`

For more information, see Assert.

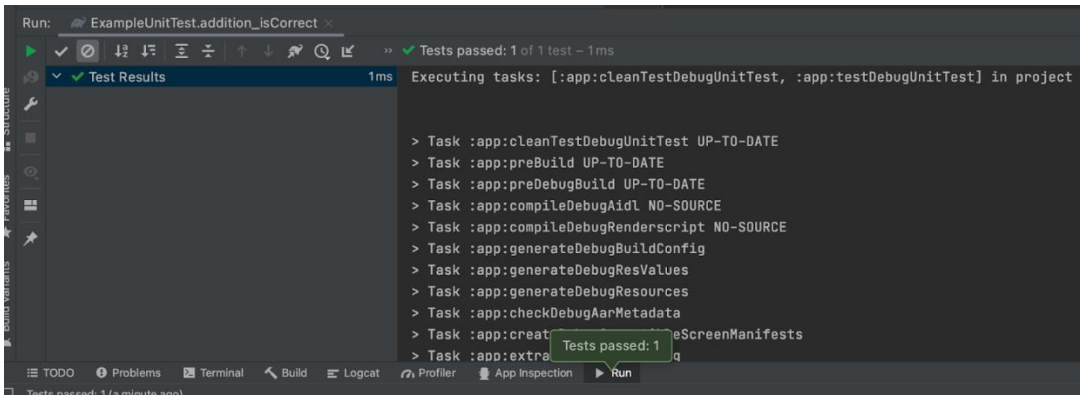
Istnieje wiele sposobów uruchamiania testów w Android Studio, do których zagłębisz się później. Teraz zachowaj prostotę.

1. Kliknij strzałki obok `addition_isCorrect`deklaracji metody, a następnie wybierz opcję **Uruchom „ExampleUnitTest.addition_isCorrect”** .

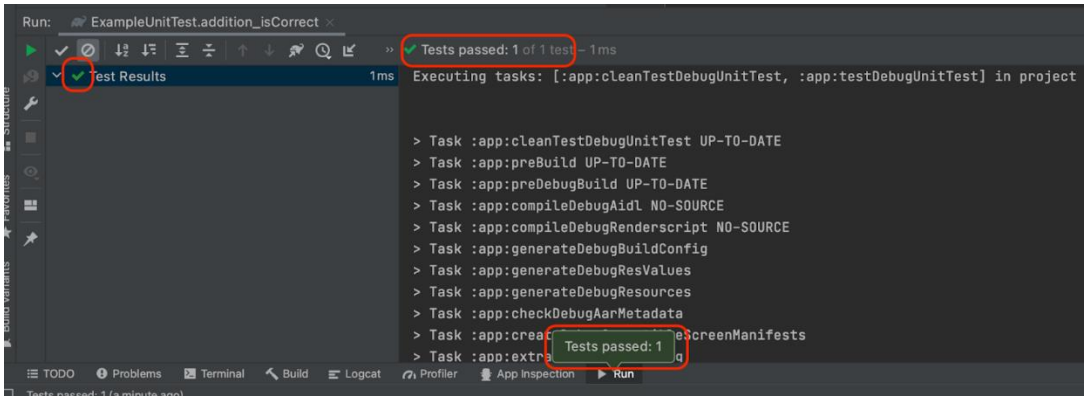


To jest określane jako pozytywny test. Innymi słowy, twierdzenie jest twierdzące. 2 + 2 równa się 4. Alternatywnie, moglibyśmy napisać test negatywny, w którym twierdzenie jest negatywne. Na przykład: 2 + 2 nie jest równe 5.

W okienku **Uruchom** powinieneś zobaczyć coś takiego jak ten zrzut ekranu:



Wiele wskazuje na to, że test się powiódł, a mianowicie zielone kontrole i liczba pomyślnych testów.

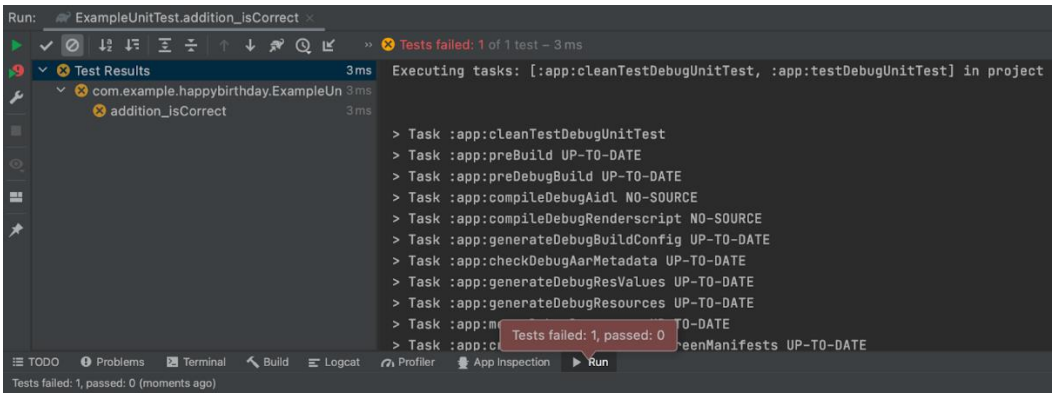


1. Zmodyfikuj test, aby zobaczyć, jak wygląda awaria. Zmień `2 + 2` na `2 + 3`, a następnie ponownie wykonaj test. Pamiętaj, że eksperymentujesz tylko z wygenerowanym kodem, aby zdobyć doświadczenie w działaniu testów. Te zmiany nie mają żadnego związku z funkcją Dice Roller.

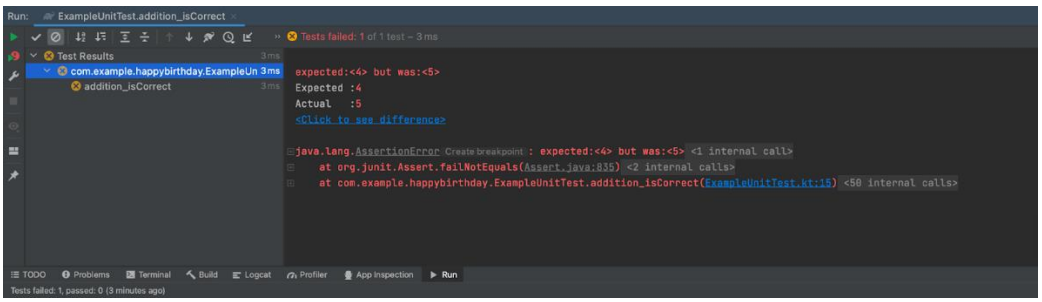
`ExampleUnitTest.kt`

```
class ExampleUnitTest {
  @Test
  fun addition_isCorrect() {
     assertEquals(4, 2 + 3)
  }
}
```

Po uruchomieniu reszty powinieneś zobaczyć coś takiego jak ten zrzut ekranu:



Czerwony tekst oznacza niepowodzenie testu. Kliknięcie pozycji w menu wyników testu powoduje wyświetlenie komunikatu o błędzie wskazującego, dlaczego test się nie powiódł.



W tym przypadku komunikat wskazuje, że potwierdzenie nie powiodło się, ponieważ oczekiwało wyniku `4`, ale rzeczywista wartość to `5`. Ma to sens, ponieważ zmieniłeś rzeczywistą wartość na `2 + 3`, ale oczekiwaną wartość pozostawiłeś jako `4`. Możesz także zobaczyć linię, na której test się nie powiódł. W tym przypadku jest to linia 15, oznaczona jako `ExampleUnitTest.kt:15`.

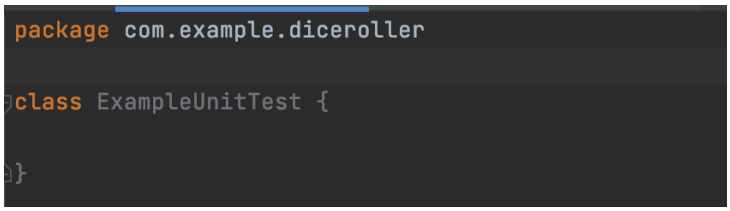
1. Ze względu na dokładność zmień oczekiwaną wartość z `4`na `5`i uruchom test ponownie. Test powinien teraz przejść, ponieważ oczekiwana wartość pasuje do rzeczywistego wyniku danego kodu.

3. Napisz swój pierwszy test jednostkowy

Teraz, gdy nabrałeś trochę komfortu dzięki testom jednostkowym, możesz napisać własny test jednostkowy, który będzie bardziej odpowiedni dla aplikacji Dice Roller.

Jak już zauważyłeś, podstawowa funkcjonalność aplikacji Dice Roller opiera się na generatorze liczb losowych. Niestety generatory liczb losowych są bardzo trudne do przetestowania, ponieważ nie można być pewnym wyniku losowo wygenerowanej liczby. Celem tego testu jest upewnienie się, że kiedy rzucisz kostką lub wywołasz `roll`metodę na `dice`klasie, otrzymasz odpowiednią liczbę z powrotem. Test, który piszesz, po prostu sprawdza, czy wyjściem generatora liczb losowych jest liczba z zakresu określonego dla generatora.

1. W `ExampleUnitTest.kt`pliku usuń wygenerowaną metodę testową i zaimportuj instrukcje. Twój plik powinien teraz wyglądać tak:



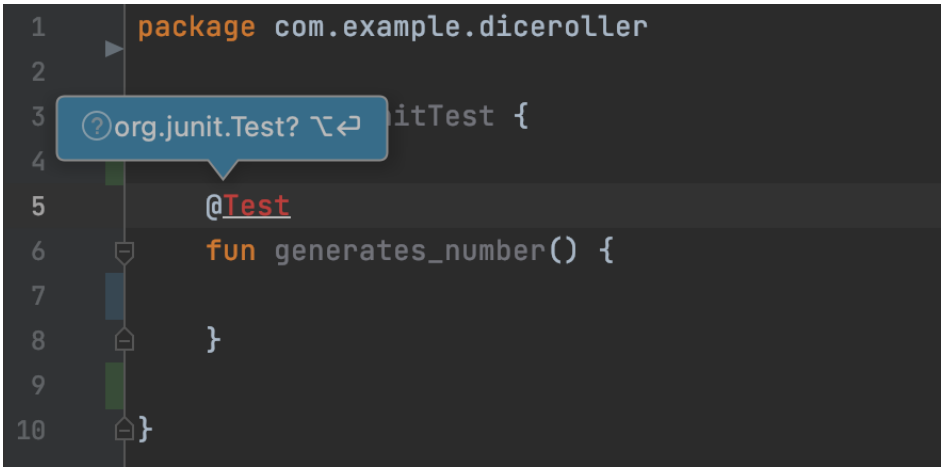
1. Utwórz `generates_number()` funkcję:

`ExampleUnitTest.kt`

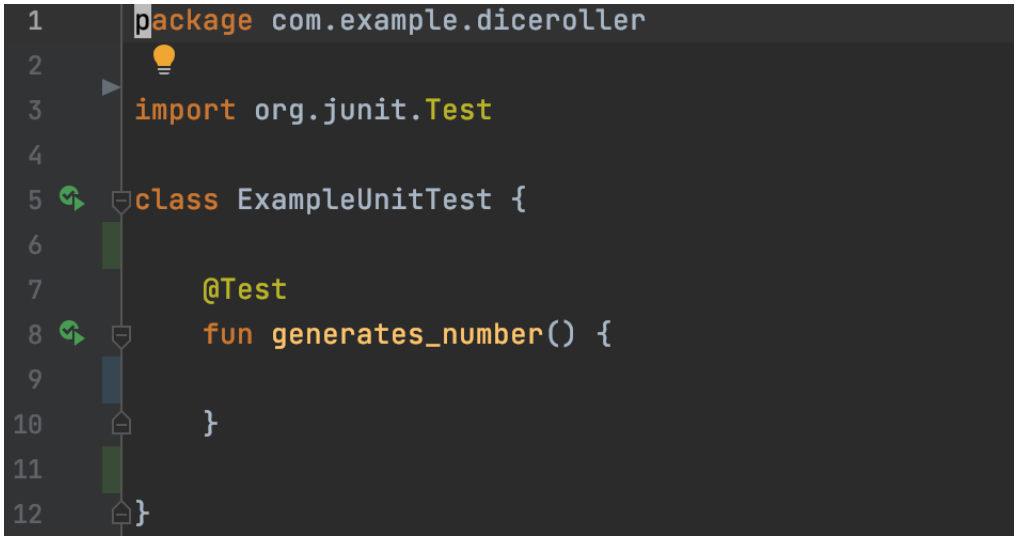
```
fun generates_number() {
}
```

1. Opisz `generates_number()` metodę `@Test` adnotacją. Zauważ, że kiedy próbujesz wywołać `@Test`, tekst jest czerwony. Dzieje się tak, ponieważ nie może znaleźć deklaracji tej adnotacji, więc musisz ją zaimportować. Możesz to zrobić automatycznie po naciśnięciu `Control+Enter` (lub `Options+Return` na Macu).

Jeśli klikniesz wiersz kodu, powinieneś zobaczyć monit o zaimportowanie:



Alternatywnie możesz również skopiować i wkleić `import org.junit.Test` plik po nazwie pakietu, ale przed deklaracją klasy. Kod powinien teraz wyglądać tak:



1. Utwórz instancję `Dice`obiektu.

`ExampleUnitTest.kt`

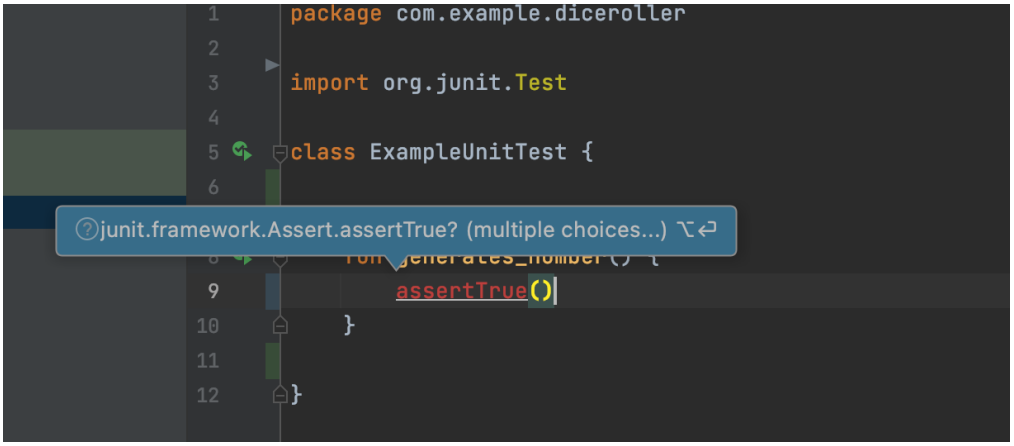
```
@Test
fun generates_number() {
  val dice = Dice(6)
}
```

1. Następnie wywołaj `roll()`metodę w tym wystąpieniu i zapisz zwróconą wartość.

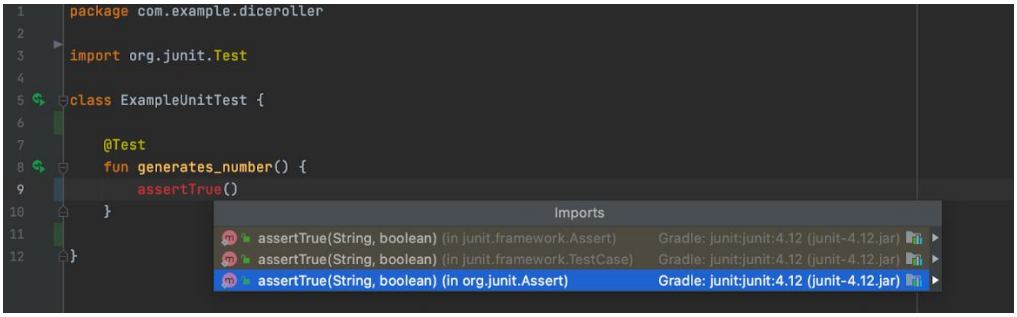
`ExampleUnitTest.kt`

```
@Test
fun generates_number() {
  val dice = Dice(6)
  val rollResult = dice.roll()
}
```

1. Na koniec dokonaj faktycznego stwierdzenia. Innymi słowy, musisz zapewnić, że metoda zwróciła wartość mieszczącą się w liczbie stron, które przekazałeś. W tym przypadku wartość musi być większa niż `0`i mniejsza niż `7`. Aby to osiągnąć, użyj `assertTrue()`metody. Zauważ, że kiedy próbujesz wywołać `assertTrue()`metodę, tekst jest na początku czerwony. Dzieje się tak, ponieważ nie może znaleźć deklaracji tej metody, więc musisz ją zaimportować, podobnie jak w przypadku adnotacji.



Możesz go automatycznie zaimportować, jak omówiono wcześniej. Zauważ jednak, że tym razem masz wiele opcji do wyboru. W takim przypadku powinna to być opcja z `org.junit.Assert`pakietu:



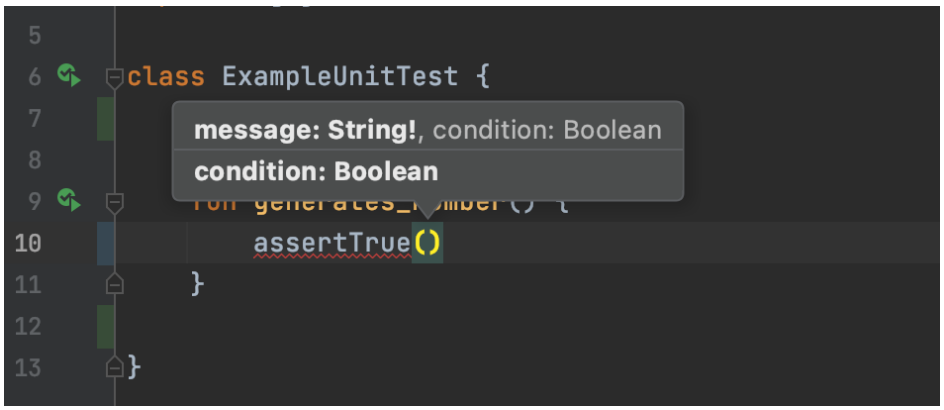
Alternatywnie możesz wkleić ten kod po instrukcji import dla adnotacji testowej:

ExampleUnitTest.kt

```
import org.junit.Assert.assertTrue
```

Twój kod wygląda teraz tak:

Jeśli umieścisz kursor między nawiasami i naciśniesz Control+P(lub Command+Pna Macu), zobaczysz podpowiedź, która pokazuje, jakie parametry przyjmuje metoda:



Metoda assertTrue()przyjmuje dwa parametry: a Stringi a Boolean. Jeśli potwierdzenie nie powiedzie się, ciąg jest komunikatem wyświetlanym w konsoli. Wartość logiczna jest instrukcją warunkową. Ustaw wiadomość na:

ExampleUnitTest.kt

```
"The value of rollResult was not between 1 and 6"
```

Jak wspomniano wcześniej, testowanie liczb losowych jest wyzwaniem, ponieważ wartości liczby nie można przewidzieć ze względu na charakter jej losowości. Wszystko, co można zrobić, to upewnić się, że wartość mieści się w określonym zakresie. Ustaw parametr warunku na:

ExampleUnitTest.kt

rollResult in 1..6

Kod powinien wyglądać tak:

ExampleUnitTest.kt

```
@Test
fun generates_number() {
  val dice = Dice(6)
  val rollResult = dice.roll()
  assertTrue("The value of rollResult was not between 1 and 6", rollResult in 1..6)
}
```

1. Kliknij strzałki obok funkcji, a następnie wybierz opcję **Uruchom 'ExampleUnitTest.generates_number()'** .

Jeśli Twój kod wygląda jak poprzedni fragment kodu, Twój test powinien zaliczyć!

1. Opcjonalnie: Aby uzyskać dodatkową praktykę, zmodyfikuj kości tak, aby były 4- lub 5-stronne bez zmiany asercji, aby test zakończył się niepowodzeniem.

4. Gratulacje

Nauczyłeś się:

- Znaczenie testowania.
- Jak wygląda test jednostkowy.
- Jak przeprowadzić test jednostkowy.
- Pewna typowa składnia testowania.
- Jak napisać test jednostkowy.

Ucz się więcej

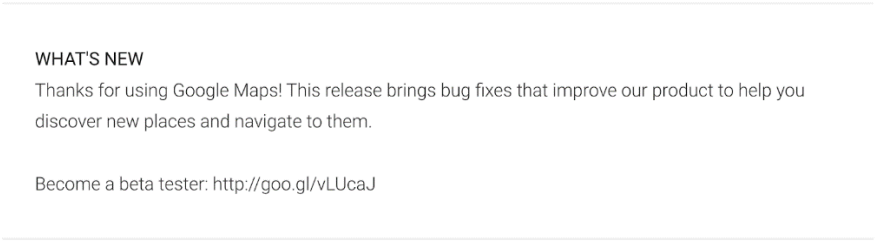
- Testuj aplikacje na Androida

Wprowadzenie do debugowania

1. Zanim zaczniesz

Każdy, kto korzystał z oprogramowania, najprawdopodobniej napotkał **błąd** . Błąd to błąd w oprogramowaniu, który powoduje niezamierzone zachowanie, takie jak awaria aplikacji lub funkcja nie działająca zgodnie z oczekiwaniami. Wszyscy programiści, niezależnie od

doświadczenia, wprowadzają błędy podczas pisania kodu, a jedną z najważniejszych umiejętności programisty Androida jest ich identyfikacja i naprawa. Nierzadko można zobaczyć całe wydania aplikacji poświęcone naprawianiu błędów. Zobacz na przykład szczegóły wersji Map Google poniżej:



Proces naprawiania błędów nazywa się **debugowaniem** . Słynny informatyk Brian Kernighan powiedział kiedyś, że „najskuteczniejszym narzędziem do debugowania jest nadal staranne przemyślenie, w połączeniu z rozsądnie umieszczonymi oświadczeniami drukowanymi”. Chociaż być może znasz już instrukcję println() Kotlina z poprzednich laboratoriów kodowania, profesjonalni programiści Androida używają rejestrowania, aby lepiej organizować wyniki swoich programów. W tym ćwiczeniu z kodowania dowiesz się, jak korzystać z logowania w Android Studio i jak można go używać jako narzędzia do debugowania. Dowiesz się również, jak czytać dzienniki komunikatów o błędach, zwane śladami stosu, aby identyfikować i rozwiązywać błędy. Na koniec dowiesz się, jak samodzielnie badać błędy, a także jak przechwytywać dane wyjściowe z emulatora Androida w postaci zrzutu ekranu lub GIF-a uruchomionej aplikacji.

Warunki wstępne

- Wiesz, jak poruszać się po projekcie w Android Studio.

Czego się nauczysz

Pod koniec tego ćwiczenia z programowania będziesz w stanie

- Zapisuj dzienniki używając `android.util.Logger`.
- Wiedz, kiedy używać różnych poziomów dziennika.
- Używaj dzienników jako prostego i wydajnego narzędzia do debugowania.
- Jak znaleźć istotne informacje w śladzie stosu.
- Wyszukaj komunikaty o błędach, aby rozwiązać awarie aplikacji.
- Przechwytuj zrzuty ekranu i animowane pliki GIF z emulatora Androida.

Co będziesz potrzebował

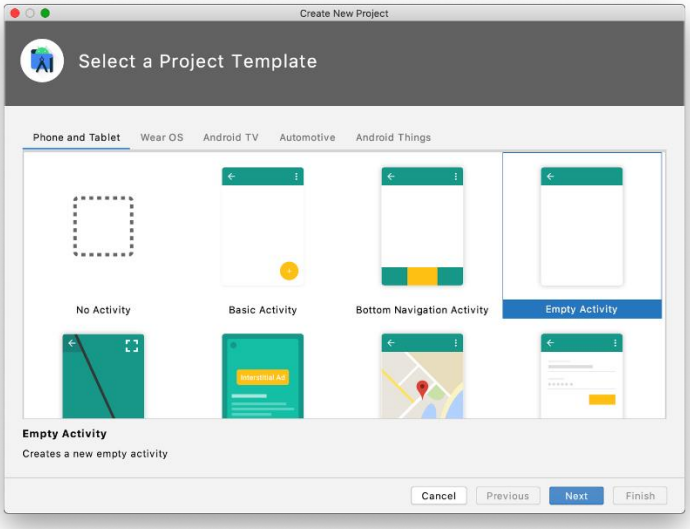
- Komputer z zainstalowanym Android Studio.

2. Utwórz nowy projekt

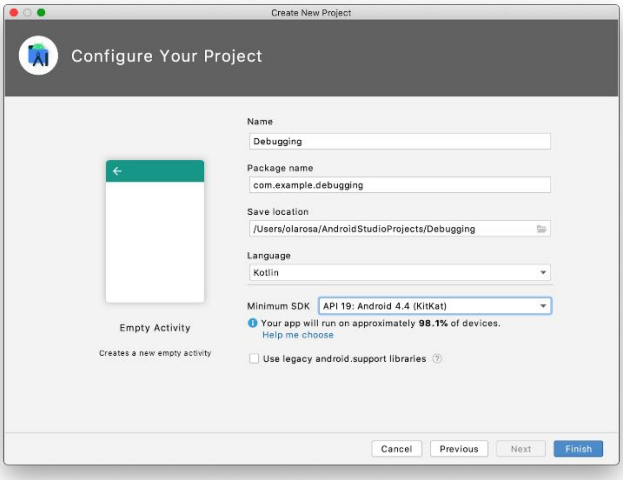
Zamiast używać dużej i złożonej aplikacji, zaczniemy od pustego projektu, aby zademonstrować instrukcje dziennika i ich użycie do debugowania.

Zacznij od utworzenia nowego projektu Android Studio, jak pokazano.

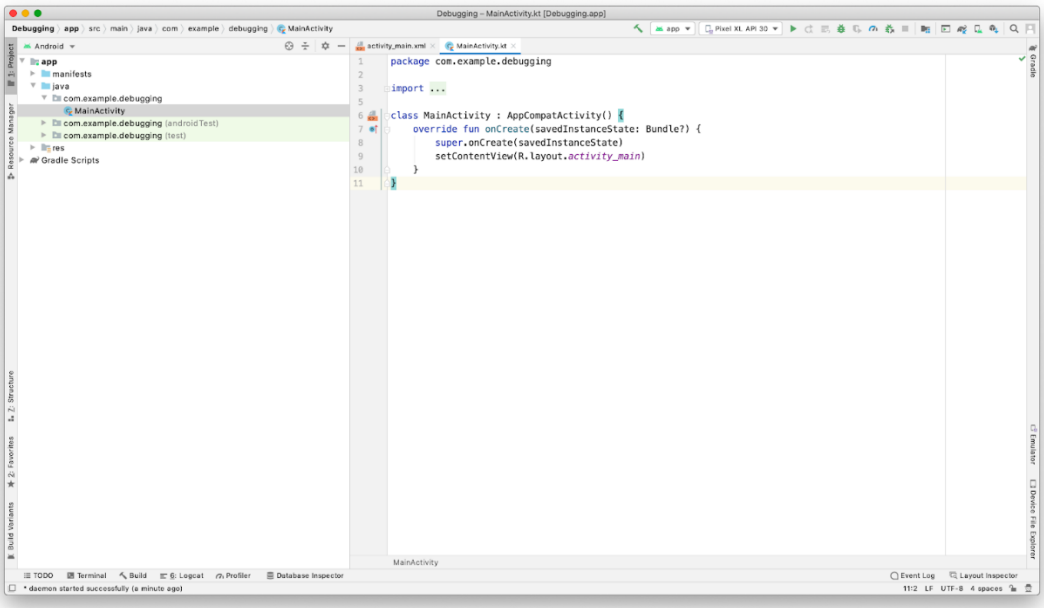
1. Na ekranie **Wybierz szablon projektu** wybierz **Puste działanie** .



1. Nazwij aplikację **Debugowanie** . Upewnij się, że język jest ustawiony na Kotlin i pozostaw wszystko inne bez zmian.



Po utworzeniu projektu zostaniesz powitany nowym projektem Android Studio, pokazującym plik o nazwie `MainActivity.kt`.



3. Wyjście logowania i debugowania

W poprzednich lekcjach używałeś `println()`oświadczenia Kotlina do generowania tekstu wyjściowego. W aplikacji na Androida najlepszym rozwiązaniem dla rejestrowania danych wyjściowych jest użycie klasy Log . Istnieje kilka funkcji rejestrowania danych wyjściowych w postaci `Log.v()`, `Log.d()`, `Log.i()`, `Log.w()`lub `Log.e()`. Metody te przyjmują dwa parametry: pierwszy, zwany „tagiem”, jest ciągiem znaków identyfikującym źródło komunikatu dziennika (takim jak nazwa klasy, która zarejestrowała tekst). Drugi to aktualny komunikat dziennika.

Wykonaj następujące kroki, aby rozpocząć logowanie w pustym projekcie.

1. W `MainActivity.kt`, przed deklaracją klasy, dodaj stałą o nazwie `TAG`i ustaw jej wartość na nazwę klasy, `MainActivity`.

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

Uwaga: tag dziennika jest zwykle deklarowany poza klasą. Mimo że ta zmienna jest zadeklarowana poza `MainActivity`, jest zadeklarowana jako prywatna, więc będzie dostępna tylko w `MainActivity.kt`. Oznacza to, że możesz również zadeklarować `TAG`zmienną dla innych klas. Aby dowiedzieć się więcej, zapoznaj się z dokumentacją Kotlina dotyczącą modyfikatorów widoczności .

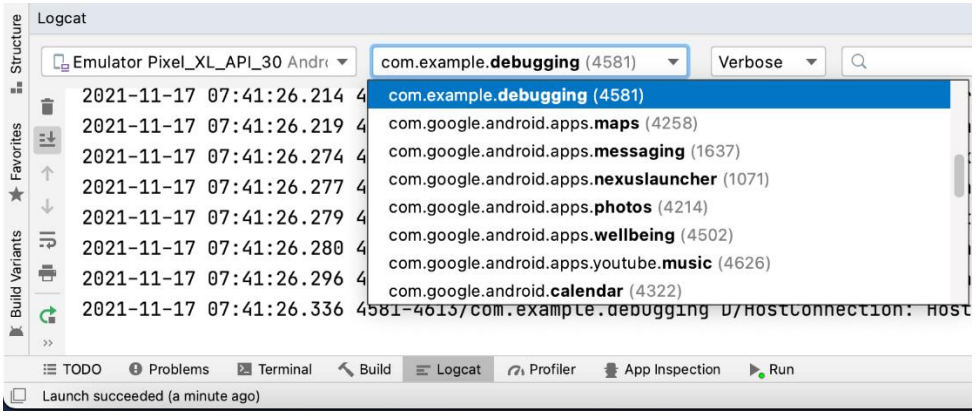
1. Dodaj nową funkcję do `MainActivity`klasy o nazwie `logging()`, jak pokazano.

```
fun logging() {
  Log.v(TAG, "Hello, world!")
}
```

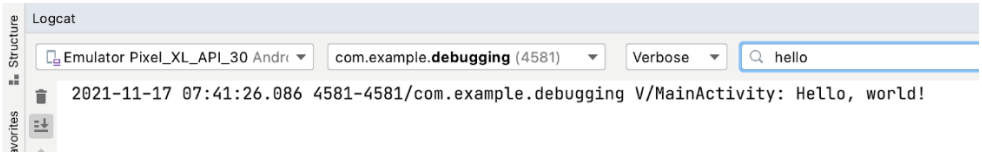
1. Zadzwoń . `logging()`_ `onCreate()`Nowa `onCreate()`metoda powinna wyglądać następująco.

```
override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   setContentView(R.layout.activity_main)
   logging()
}
```

1. Uruchom aplikację, aby zobaczyć dzienniki w akcji. Dzienniki pojawiają się w oknie Logcat na dole ekranu. Ponieważ Logcat pokaże dane wyjściowe z innych procesów na urządzeniu (lub emulatorze), możesz wybrać swoją aplikację (com.example.debugging) z menu rozwijanego, aby odfiltrować wszelkie dzienniki, które nie są istotne dla Twojej aplikacji.



W oknie danych wyjściowych powinieneś być w stanie zobaczyć swoje "Hello, world!" wyjście. W razie potrzeby wpisz „hello” w polu wyszukiwania u góry okna Logcat, aby przeszukać wszystkie dzienniki.



Poziomy dziennika

Powodem istnienia różnych funkcji dziennika, nazwanych różnymi literami, jest to, że odpowiadają one różnym poziomom dziennika. W zależności od rodzaju informacji, które chcesz wyprowadzić, użyjesz innego poziomu dziennika, aby pomóc w filtrowaniu ich w danych `Logcat`wyjściowych. Istnieje pięć głównych poziomów dziennika, z których będziesz korzystać regularnie.

Poziom dziennika	Przypadek użycia	Przykład
BŁĄD	Dzienniki BŁĘDÓW zgłaszają, że coś poszło poważnie nie tak, na przykład przyczyna awarii aplikacji.	Log.e(TAG, "The cake was left in t for too long and burned.").
OSTRZEGAĆ	Dzienniki WARN są mniej poważne niż błąd, ale nadal zgłaszają coś, co należy naprawić, aby uniknąć poważniejszych błędów. Przykładem może być wywołanie funkcji, która	Log.w(TAG, "This oven does not h evenly. You may want to turn the

	jest *przestarzała* , co oznacza, że jej użycie jest odradzane na rzecz nowszej alternatywy.	around halfway through to promo browning.")
INFORMACJE	Dzienniki INFO dostarczają przydatnych informacji, takich jak pomyślne zakończenie operacji.	Log.i(TAG, "The cake is ready to b served.").println("The cake has co
ODPLUSKWIĆ	Dzienniki DEBUG zawierają informacje, które mogą być przydatne podczas badania problemu. Te dzienniki nie są obecne w kompilacjach wydań, takich jak ta, którą publikujesz w Sklepie Google Play.	Log.d(TAG, "Cake was removed fr oven after 55 minutes. Recipe call the cake to be removed after 50 - minutes.")
GADATLIWY	Jak sama nazwa wskazuje, verbose to najmniej określony poziom dziennika. To, co jest uważane za dziennik debugowania, a nie pełny, jest nieco subiektywne, ale ogólnie pełny dziennik to coś, co można usunąć po zaimplementowaniu funkcji, podczas gdy dziennik debugowania może nadal być przydatny do debugowania. Te dzienniki również nie są uwzględniane w kompilacjach wydania.	Log.v(TAG, "Put the mixing bowl counter.")Log.v(TAG, "Grabbed th from the refrigerator.")Log.v(TAG "Plugged in the stand mixer.")

Należy pamiętać, że nie ma ustalonych reguł określających, kiedy używać każdego typu poziomu dziennika, a zwłaszcza kiedy używać `DEBUG`i `VERBOSE`. Zespoły programistyczne mogą tworzyć własne wytyczne dotyczące tego, kiedy używać każdego poziomu rejestrowania, lub może zdecydować, że `VERBOSE`w ogóle nie będą używać niektórych poziomów rejestrowania, takich jak . Ważną rzeczą do zapamiętania na temat tych dwóch poziomów dziennika jest to, że nie są one obecne w kompilacjach wydania, więc używanie dzienników do debugowania nie wpłynie na wydajność opublikowanych aplikacji, podczas gdy `println()`instrukcje pozostają w kompilacjach wydania i mają negatywny wpływ na wydajność.

Ciekawostka: W rzeczywistości istnieje inny poziom dziennika, Log.wtf() (Co za straszna porażka), który istnieje do rejestrowania rzeczy, co do których masz pewność, że nigdy się nie pojawią, stąd nazwa. Te dzienniki są `ASSERT`na poziomie, o poziom powyżej `ERROR`.

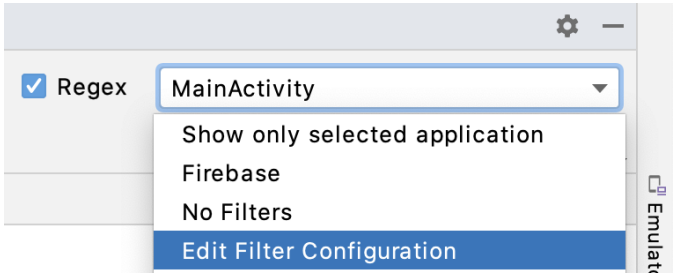
Zobaczmy, jak wyglądają te różne poziomy dzienników w Logcat.

1. W `MainActivity.kt`programie zastąp zawartość `logging()`metody następującą.

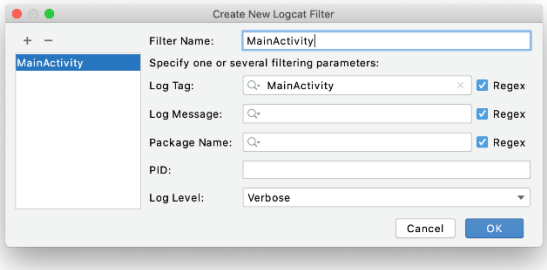
```
fun logging() {
  Log.e(TAG, "ERROR: a serious error like an app crash")
  Log.w(TAG, "WARN: warns about the potential for serious errors")
  Log.i(TAG, "INFO: reporting technical information, such as an operation succeeding")
  Log.d(TAG, "DEBUG: reporting technical information useful for debugging")
  Log.v(TAG, "VERBOSE: more verbose than DEBUG logs")
}
```

Uwaga: tag dziennika jest zwykle deklarowany poza klasą. Mimo że ta zmienna jest zadeklarowana poza `MainActivity`, jest zadeklarowana jako prywatna, więc będzie dostępna tylko w `MainActivity.kt`. Oznacza to, że możesz również zadeklarować `TAG`zmienną dla innych klas. Aby dowiedzieć się więcej, zapoznaj się z dokumentacją Kotlina dotyczącą modyfikatorów widoczności .

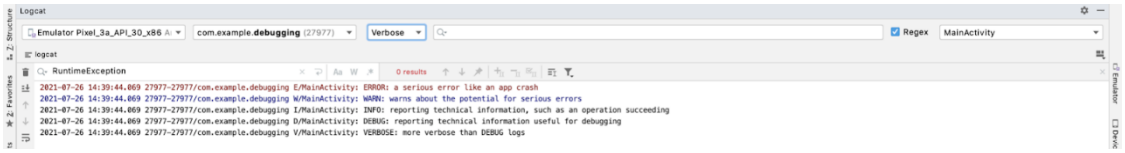
1. Uruchom aplikację i obserwuj dane wyjściowe w Logcat. W razie potrzeby przefiltruj dane wyjściowe, aby wyświetlić tylko dzienniki z procesu **com.example.debugging** . Możesz także filtrować dane wyjściowe, aby wyświetlać tylko logi z tagiem „MainActivity”. Aby to zrobić, wybierz **Edytuj konfigurację filtra** z menu rozwijanego w prawym górnym rogu okna Logcat.



1. Następnie wpisz „MainActivity” dla **tagu dziennika** i utwórz nazwę filtra, jak pokazano.

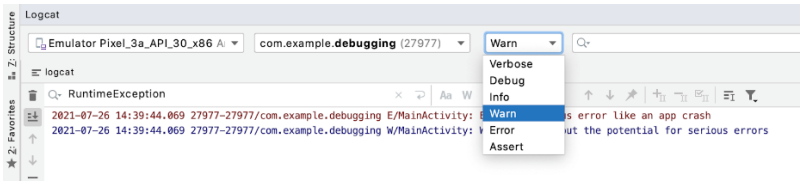


1. Teraz powinieneś widzieć tylko komunikaty dziennika z tagiem „MainActivity”.



Zwróć uwagę, że przed nazwą klasy znajduje się litera, na przykład `W/MainActivity`, odpowiadająca poziomowi rejestrowania. Ponadto `WARN`dziennik jest wyświetlany na niebiesko, podczas gdy `ERROR`dziennik jest wyświetlany na czerwono, podobnie jak błąd krytyczny w poprzednim przykładzie.

1. Tak jak można filtrować dane wyjściowe debugowania według procesu, można również filtrować dane wyjściowe według poziomu dziennika. Domyślnie jest to **opcja Verbose,** która wyświetla `VERBOSE`dzienniki i wyższe poziomy dzienników. Wybierz **Ostrzegaj** z menu rozwijanego i zauważ, że wyświetlane są tylko dzienniki teraz `WARN`i poziomy.`ERROR`



1. Ponownie zmień menu rozwijane na **Assert** i zwróć uwagę, że nie są wyświetlane żadne dzienniki. To odfiltrowuje wszystko, co jest na `ERROR`poziomie i poniżej.

Chociaż może to wydawać się traktowanie `println()`oświadczeń trochę zbyt poważnie, podczas tworzenia większych aplikacji będzie o wiele więcej danych wyjściowych Logcat, a używanie różnych poziomów dziennika pozwoli Ci wybrać najbardziej przydatne i istotne informacje. Korzystanie z dziennika jest uważane za najlepsze rozwiązanie i jest preferowane `println()`w przypadku opracowywania systemu Android, ponieważ dzienniki debugowania i pełne nie mają wpływu na wydajność w kompilacjach wersji. Możesz także filtrować dzienniki na podstawie różnych poziomów dziennika. Wybranie prawidłowego poziomu dziennika przyniesie korzyści innym członkom zespołu programistów, którzy mogą nie być tak zaznajomieni z kodem jak Ty, i znacznie ułatwi identyfikowanie i rozwiązywanie błędów.

4. Logi z komunikatami o błędach

Przedstaw błąd

W pustym projekcie nie ma zbyt wiele do debugowania. Wiele błędów, które napotkasz jako programista Androida, wiąże się z awariami aplikacji – oczywiście nie jest to dobre wrażenia użytkownika. Dodajmy kod, który powoduje awarię tej aplikacji.

Być może pamiętasz, że nauczyłeś się na lekcjach matematyki, że nie możesz podzielić liczby przez zero. Zobaczmy, co się stanie, gdy spróbujemy dzielić przez zero w kodzie.

1. Dodaj następującą funkcję do `MainActivity.kt`powyższej `logging()`funkcji. Ten kod zaczyna się od dwóch liczb i służy `repeat`do rejestrowania wyniku pięciokrotnego dzielenia licznika przez mianownik. Za każdym razem, gdy kod w `repeat`bloku jest uruchamiany, wartość mianownika zmniejsza się o jeden. W piątej i ostatniej iteracji aplikacja próbuje podzielić przez zero.

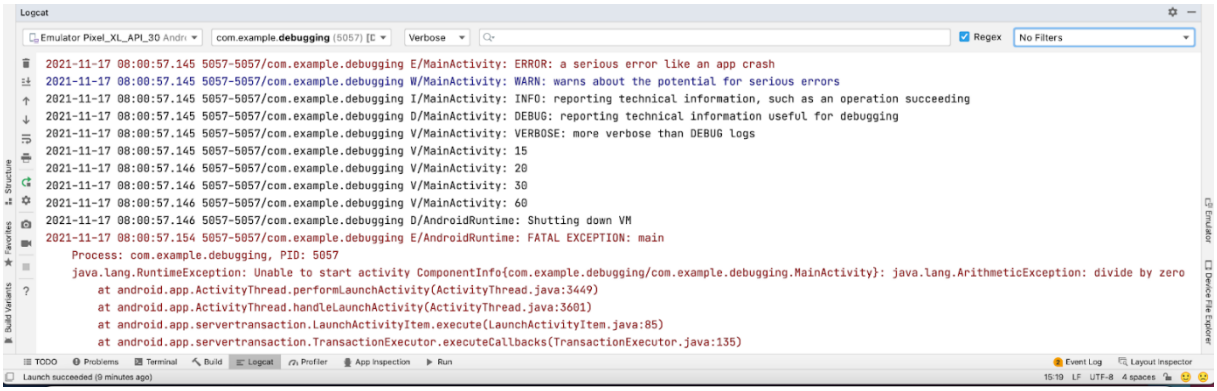
```
fun division() {
  val numerator = 60
  var denominator = 4
  repeat(5) {
    Log.v(TAG, "${numerator / denominator}")
    denominator--
  }
}
```

1. Po wywołaniu `logging()`in `onCreate()`dodaj wywołanie do `division()`funkcji.

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContentView(R.layout.activity_main)
  logging()
```

```
    division()
}
```

1. Uruchom ponownie aplikację i zauważ, że się zawiesza. Jeśli przewiniesz w dół do dzienników z `MainActivity.kt`klasy, zobaczysz `logging()`dzienniki z funkcji, którą zdefiniowałeś wcześniej, pełne dzienniki z `division()`funkcji, a następnie czerwony dziennik błędów wyjaśniający, dlaczego aplikacja uległa awarii.



Ostrzeżenie: ponieważ są one usuwane z kompilacji wydań, należy unikać wprowadzania efektów ubocznych (modyfikowania wartości w kodzie) z instrukcji dziennika. Na przykład, jeśli zmodyfikujesz powyższy przykład, aby zmniejszyć mianownik w instrukcji dziennika, tak jak w `Log.v(TAG, "${numerator / denominator--}")`, kod nadal będzie działał zgodnie z oczekiwaniami w kompilacjach debugowania. Jednak ponieważ dziennik jest usuwany w kompilacji wydania, pętla będzie kontynuowana w nieskończoność, ponieważ mianownik nie zostanie zmniejszony.

Anatomia śladu stosu

Dziennik błędów opisujący awarię (nazywany również **wyjątkiem**) nazywany jest śladem stosu. Ślad stosu pokazuje wszystkie wywołane funkcje prowadzące do wyjątku, zaczynając od ostatnio wywołanego. Pełne dane wyjściowe pokazano poniżej.

```
Process: com.example.debugging, PID: 14581
    java.lang.RuntimeException: Unable to start activity
ComponentInfo{com.example.debugging/com.example.debugging.MainActivity}:
java.lang.ArithmeticException: divide by zero
        at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3449)
        at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3601)
        at android.app.servertransaction.LaunchActivityItem.execute(LaunchActivityItem.java:85)
        at
android.app.servertransaction.TransactionExecutor.executeCallbacks(TransactionExecutor.java:135
)
        at android.app.servertransaction.TransactionExecutor.execute(TransactionExecutor.java:95)
        at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2066)
        at android.os.Handler.dispatchMessage(Handler.java:106)
        at android.os.Looper.loop(Looper.java:223)
        at android.app.ActivityThread.main(ActivityThread.java:7656)
        at java.lang.reflect.Method.invoke(Native Method)
```

```
    at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:592)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:947)
  Caused by: java.lang.ArithmeticException: divide by zero
    at com.example.debugging.MainActivity.division(MainActivity.kt:21)
    at com.example.debugging.MainActivity.onCreate(MainActivity.kt:14)
    at android.app.Activity.performCreate(Activity.java:8000)
    at android.app.Activity.performCreate(Activity.java:7984)
    at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1309)
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3422)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3601)
    at android.app.servertransaction.LaunchActivityItem.execute(LaunchActivityItem.java:85)
    at
android.app.servertransaction.TransactionExecutor.executeCallbacks(TransactionExecutor.java:135
)
    at android.app.servertransaction.TransactionExecutor.execute(TransactionExecutor.java:95)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2066)
    at android.os.Handler.dispatchMessage(Handler.java:106)
    at android.os.Looper.loop(Looper.java:223)
    at android.app.ActivityThread.main(ActivityThread.java:7656)
    at java.lang.reflect.Method.invoke(Native Method)
    at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:592)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:947)
```

To dużo tekstu! Na szczęście zazwyczaj potrzebujesz tylko kilku elementów, aby zawęzić dokładny błąd. Zacznijmy od góry.

1. `java.lang.RuntimeException:`

```
java.lang.RuntimeException: Unable to start activity
ComponentInfo{com.example.debugging/com.example.debugging.MainActivity}:
java.lang.ArithmeticException: divide by zero
```

Pierwsza linia informuje, że aplikacja nie mogła rozpocząć aktywności, co jest przyczyną awarii aplikacji. Kolejny wiersz zawiera nieco więcej informacji. W szczególności przyczyną niemożności rozpoczęcia działania był `ArithmeticException`. Dokładniej, typ `ArithmeticException`„podziel przez zero”.

1. `Caused by:`

```
Caused by: java.lang.ArithmeticException: divide by zero
    at com.example.debugging.MainActivity.division(MainActivity.kt:21)
```

Jeśli przewiniesz w dół do wiersza „Spowodowane przez”, ponownie wyświetli się komunikat o błędzie „dzielenia przez zero”. Tym razem pokazuje również dokładną funkcję, w której wystąpił błąd (`division()`) oraz dokładny numer wiersza (21). Nazwa pliku i numer wiersza w oknie Logcat są hiperłączem. Dane wyjściowe zawierają również nazwę funkcji, w której wystąpił błąd `division()`, oraz funkcję, która go wywołała, `onCreate()`.

Nic z tego nie powinno dziwić, ponieważ błąd został celowo wprowadzony. Jeśli jednak musisz określić przyczynę nieznanego błędu, znajomość dokładnego typu wyjątku, nazwy funkcji i numeru wiersza dostarcza niezwykle przydatnych informacji.

Dlaczego „ślad stosu”?

Termin „ścieżka stosu” może brzmieć jak dziwny termin dla tekstu wynikowego z błędu. Aby lepiej zrozumieć, jak to działa, musisz wiedzieć trochę więcej o stosie funkcji.

Gdy jedna funkcja wywołuje inną funkcję, urządzenie nie uruchomi żadnego kodu z pierwszej funkcji, dopóki druga funkcja nie zakończy. Gdy druga funkcja zakończy wykonywanie, pierwsza funkcja wznawia działanie w miejscu, w którym została przerwana. To samo dotyczy wszystkich funkcji wywoływanych przez drugą funkcję. Druga funkcja nie wznowi wykonywania, dopóki trzecia funkcja (i inne wywołane przez nią funkcje) nie zakończy działania, a pierwsza funkcja nie zostanie wznowiona, dopóki druga funkcja nie zakończy wykonywania. Jest to podobne do stosu w świecie fizycznym, takiego jak stos talerzy lub stos kart. Jeśli chcesz wziąć talerz, weźmiesz górny. Nie da się zdjąć talerza niżej w stosie bez uprzedniego usunięcia wszystkich talerzy znajdujących się nad nim.

Stos funkcji można zilustrować następującym kodem.

```
val TAG = ...

fun first() {
  second()
  Log.v(TAG, "1")
}

fun second() {
  third()
  Log.v(TAG, "2")
  fourth()
}

fun third() {
  Log.v(TAG, "3")
}

fun fourth() {
  Log.v(TAG, "4")
}
```

Jeśli zadzwonisz `first()`, numery zostaną zarejestrowane w następującej kolejności.

```
3
2
4
1
```

Dlaczego to? Po wywołaniu pierwszej funkcji natychmiast wywołuje `second()`, więc numer `1`nie może być od razu zalogowany. Stos funkcji wygląda tak.

```
second()
first()
```

Druga funkcja następnie wywołuje `third()`, co dodaje ją do stosu funkcji.

```
third()
second()
first()
```

Trzecia funkcja wypisuje następnie liczbę `3`. Po zakończeniu wykonywania jest usuwany ze stosu funkcji.

```
second()
first()
```

Funkcja `second()`następnie rejestruje numer `2`, a następnie wywołuje `fourth()`. Do tej pory liczby `3`, a następnie `2`, zostały zarejestrowane, a stos funkcji wygląda teraz następująco.

```
fourth()
second()
first()
```

Funkcja `fourth()`drukuje numer `4`i jest usuwana (wyskakiwana) ze stosu funkcji. Funkcja `second()`następnie kończy wykonywanie i jest zdejmowana ze stosu funkcji. Teraz, gdy `second()`wszystkie wywołane funkcje zostały zakończone, urządzenie wykonuje pozostały kod, w `first()`którym wypisuje liczbę `1`.

W ten sposób liczby są rejestrowane w kolejności: `4`, `2`, `3`, `1`.

Nie spiesząc się i przechodząc przez kod oraz zachowując mentalny obraz stosu funkcji, możesz dokładnie zobaczyć, jaki kod jest wykonywany iw jakiej kolejności. Samo to może być potężną techniką debugowania błędów, takich jak przykład dzielenia przez zero powyżej. Przechodzenie przez kod może również dać dobry pomysł, gdzie umieścić instrukcje dziennika, aby pomóc w debugowaniu bardziej złożonych problemów.

Uwaga: w praktyce aplikacja może technicznie uruchomić więcej niż jedną funkcję Kotlina naraz. Prawdopodobnie napotkałeś już aplikacje, które działają w tle, na przykład pobierają aplikację, gdy kontynuujesz przeglądanie sklepu Google Play. Jest to możliwe dzięki czemuś, co nazywa się *wątkami* . Dzięki wielu wątkom możesz wykonać wiele sekwencji kodu Kotlin jednocześnie. Jednak każdy wątek ma swój własny stos funkcji i działa tak jak w powyższym przykładzie. Korzystanie z wielu wątków (zwanych wielowątkowością) jest tematem zaawansowanym i wykracza poza zakres tego ćwiczenia z kodowania, ale dowiesz się więcej o wykonywaniu operacji w tle, gdy poznasz współprogramy Kotlin w części 4.

5. Używanie dzienników do identyfikacji i naprawy błędu

W poprzedniej sekcji zbadałeś ślad stosu, a konkretnie ten wiersz.

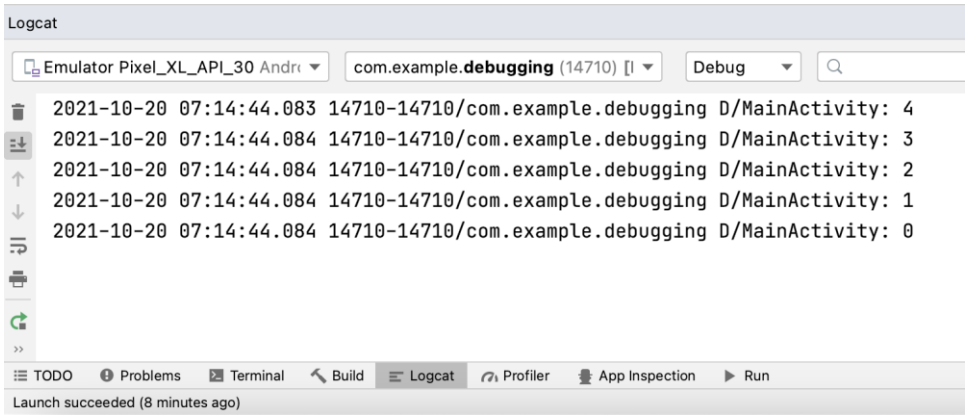
```
Caused by: java.lang.ArithmeticException: divide by zero
    at com.example.debugging.MainActivity.division(MainActivity.kt:21)
```

Tutaj możesz stwierdzić, że awaria nastąpiła w linii 21 i była związana z dzieleniem przez zero. Tak więc, gdzieś przed wykonaniem tego kodu, mianownik wynosił 0. Chociaż możesz spróbować samodzielnie przejść przez kod, co może działać doskonale w przypadku takiego małego przykładu, możesz również użyć instrukcji log, aby zaoszczędzić czas, drukując wartość mianownika przed dzieleniem przez zero.

1. Przed `Log.v()`instrukcją , dodaj `Log.d()`wywołanie, które rejestruje mianownik. `Log.d()`jest używany, ponieważ ten dziennik jest przeznaczony specjalnie do debugowania i umożliwia odfiltrowanie pełnych dzienników.

```
Log.d(TAG, "$denominator")
```

1. Uruchom ponownie swoją aplikację. Chociaż nadal się zawiesza, mianownik powinien być rejestrowany kilka razy. Możesz użyć konfiguracji filtru, aby wyświetlić tylko dzienniki z `"MainActivity"`tagiem.



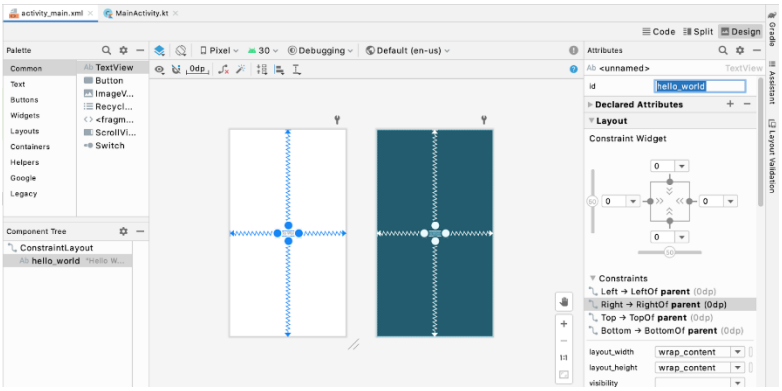
1. Widać, że drukowanych jest wiele wartości. Wygląda na to, że pętla jest wykonywana kilka razy przed awarią w piątej iteracji, gdy mianownik to `0`. Ma to sens, ponieważ mianownik to `4`, a pętla zmniejsza go o `1`dla `5`iteracji. Aby naprawić błąd, możesz zmienić liczbę iteracji w pętli z `5`na `4`. Jeśli ponownie uruchomisz aplikację, nie powinna już się zawieszać.

```
fun division() {
  val numerator = 60
  var denominator = 4
  repeat(5) {
    Log.v(TAG, "${numerator / denominator}")
    denominator--
  }
}
```

6. Przykład debugowania: dostęp do wartości, która nie istnieje

Domyślnie szablon **Puste działanie** , którego użyłeś do utworzenia projektu, dodaje pojedynczą czynność z `TextView`wyśrodkowaną na ekranie. Jak dowiedziałeś się wcześniej, możesz odwoływać się do widoków z kodu, ustawiając identyfikator w edytorze układu i uzyskując dostęp do widoku za pomocą `findViewById()`. Gdy `onCreate()`jest wywoływany w klasie aktywności, najpierw należy wywołać `setContentView()`, aby załadować plik układu (takiego jak `activity_main.xml`). Jeśli spróbujesz zadzwonić `findViewById()`przed wywołaniem `setContentView()`, aplikacja ulegnie awarii, ponieważ widok nie istnieje. Spróbujmy uzyskać dostęp do widoku, aby zilustrować inny błąd.

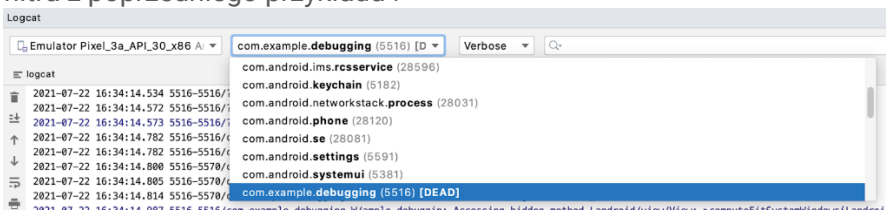
1. Otwórz `activity_main.xml`, wybierz **Witaj, świecie!** `TextView`i ustaw `id`na `hello_world`.



1. Wróć do `ActivityMain.kt`programu `onCreate()`, dodaj kod, aby pobrać `TextView`i zmień jego tekst na **„Witaj, debugowanie!”** przed wezwaniem do `setContentView()`.

```
override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   val helloTextView: TextView = findViewById(R.id.hello_world)
   helloTextView.text = "Hello, debugging!"
   setContentView(R.layout.activity_main)
   division()
}
```

1. Uruchom aplikację ponownie i zauważ, że ponownie ulega awarii natychmiast po uruchomieniu. `"MainActivity"`Aby wyświetlić logi bez tagu, może być konieczne usunięcie filtra z poprzedniego przykładu .



Wyjątkiem powinna być jedna z ostatnich rzeczy, które pojawią się w Logcat (jeśli nie, możesz wyszukać `RuntimeException`). Dane wyjściowe powinny wyglądać następująco.

```
Process: com.example.debugging, PID: 14896
   java.lang.RuntimeException: Unable to start activity
ComponentInfo{com.example.debugging/com.example.debugging.MainActivity}:
```

```
java.lang.NullPointerException: findViewById(R.id.hello_world) must not be null
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3449)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3601)
    at android.app.servertransaction.LaunchActivityItem.execute(LaunchActivityItem.java:85)
    at
android.app.servertransaction.TransactionExecutor.executeCallbacks(TransactionExecutor.java:135
)
    at android.app.servertransaction.TransactionExecutor.execute(TransactionExecutor.java:95)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2066)
    at android.os.Handler.dispatchMessage(Handler.java:106)
    at android.os.Looper.loop(Looper.java:223)
    at android.app.ActivityThread.main(ActivityThread.java:7656)
    at java.lang.reflect.Method.invoke(Native Method)
    at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:592)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:947)
  Caused by: java.lang.NullPointerException: findViewById(R.id.hello_world) must not be null
    at com.example.debugging.MainActivity.onCreate(MainActivity.kt:14)
    at android.app.Activity.performCreate(Activity.java:8000)
    at android.app.Activity.performCreate(Activity.java:7984)
    at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1309)
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3422)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3601)
    at android.app.servertransaction.LaunchActivityItem.execute(LaunchActivityItem.java:85)
    at
android.app.servertransaction.TransactionExecutor.executeCallbacks(TransactionExecutor.java:135
)
    at android.app.servertransaction.TransactionExecutor.execute(TransactionExecutor.java:95)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2066)
    at android.os.Handler.dispatchMessage(Handler.java:106)
    at android.os.Looper.loop(Looper.java:223)
    at android.app.ActivityThread.main(ActivityThread.java:7656)
    at java.lang.reflect.Method.invoke(Native Method)
    at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:592)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:947)
```

Tak jak poprzednio, na górze jest napisane „Nie można rozpocząć aktywności”. Ma to sens, ponieważ aplikacja uległa awarii przed `MainActivity`uruchomieniem. Następna linia mówi nieco więcej o błędzie.

```
java.lang.RuntimeException: Unable to start activity
ComponentInfo{com.example.debugging/com.example.debugging.MainActivity}:
java.lang.NullPointerException: findViewById(R.id.hello_world) must not be null
```

W dalszej części śladu stosu zobaczysz również ten wiersz, pokazujący dokładne wywołanie funkcji i numer wiersza.

Caused by: java.lang.NullPointerException: findViewById(R.id.hello_world) must not be null

Co dokładnie oznacza ten błąd i czym dokładnie jest wartość „null”? Chociaż jest to wymyślony przykład i możesz już mieć pomysł, dlaczego aplikacja uległa awarii, nieuchronnie natkniesz się na komunikaty o błędach, których wcześniej nie widziałeś. Kiedy tak się dzieje, prawdopodobnie nie jesteś pierwszym, który dostrzeże błąd, a nawet najbardziej doświadczeni programiści często wyświetlają komunikat o błędzie w Google, aby zobaczyć, jak inni rozwiązali problem. Wyszukiwanie tego błędu daje kilka wyników ze StackOverflow, witryny, w której programiści mogą zadawać pytania i udzielać odpowiedzi na temat błędnego kodu lub bardziej ogólnych tematów programistycznych.

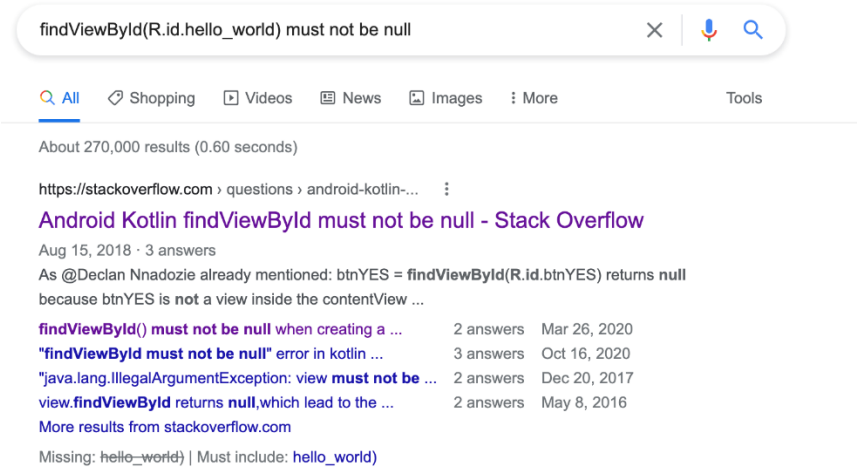
Ponieważ może być wiele pytań z podobnymi, ale nie dokładnie takimi samymi odpowiedziami, podczas samodzielnego wyszukiwania odpowiedzi pamiętaj o poniższych wskazówkach.

1. Ile lat ma odpowiedź? Odpowiedzi sprzed kilku lat mogą być już nieaktualne lub mogą dotyczyć przestarzałej wersji języka lub struktury.
2. Czy odpowiedź jest za pomocą Javy czy Kotlina? Czy Twój problem dotyczy jednego lub drugiego języka, czy też jest związany z konkretnymi ramami?
3. Odpowiedzi oznaczone jako „zaakceptowane” lub z większą liczbą głosów za mogą być wyższej jakości, ale pamiętaj, że inne odpowiedzi mogą nadal dostarczać cennych informacji.

Liczba wskazuje liczbę głosów za (lub przeciw), a zielony znacznik wyboru oznacza zaakceptowaną odpowiedź.

Jeśli nie możesz znaleźć istniejącego pytania, zawsze możesz zadać nowe pytanie. Zadając pytanie na StackOverflow (lub dowolnej witrynie), warto pamiętać o tych wskazówkach .

Śmiało i wyszukaj błąd .



Jeśli przeczytasz niektóre odpowiedzi, przekonasz się, że błąd może mieć wiele różnych przyczyn. Jednak biorąc pod uwagę, że celowo dzwoniłeś `findViewById(`wcześniej) `setContentView()`, niektóre odpowiedzi na to pytanie brzmią obiecująco. Na przykład druga najczęściej głosowana odpowiedź brzmi:

„Prawdopodobnie wywołujesz findViewById przed wywołaniem setContentView? W takim przypadku spróbuj wywołać findViewById **PO** wywołaniu setContentView”

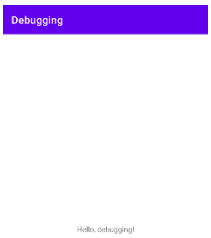
Po zobaczeniu tej odpowiedzi możesz zweryfikować w kodzie, że rzeczywiście wywołanie to `findViewById()`jest zbyt wczesne i następuje przed `setContentView()`, i że zamiast tego należy je wywołać po `setContentView()`.

Napraw błąd, aktualizując kod.

1. Przenieś połączenie do `findViewById()`i linię, która ustawia `helloTextView`tekst poniżej połączenia do `setContentView()`. Nowa `onCreate()`metoda powinna wyglądać tak, jak pokazano poniżej. Możesz także dodać dzienniki, jak pokazano, aby sprawdzić, czy błąd został naprawiony.

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContentView(R.layout.activity_main)
  Log.d(TAG, "this is where the app crashed before")
  val helloTextView: TextView = findViewById(R.id.hello_world)
  Log.d(TAG, "this should be logged if the bug is fixed")
  helloTextView.text = "Hello, debugging!"
  logging()
  division()
}
```

1. Uruchom ponownie aplikację. Pamiętaj, że aplikacja nie ulega już awarii, a tekst jest aktualizowany zgodnie z oczekiwaniami.



Zrób zrzuty ekranu

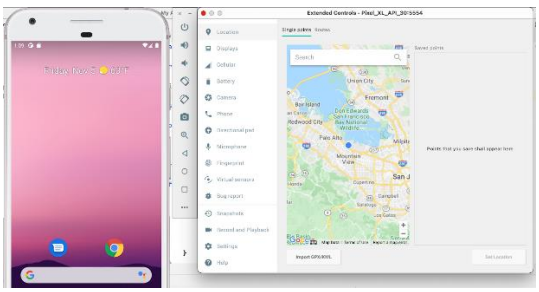
Do tej pory prawdopodobnie widziałeś w tym kursie wiele zrzutów ekranu z emulatora Androida. Robienie zrzutów ekranu jest stosunkowo proste, ale może być przydatne do dzielenia się informacjami, takimi jak kroki w celu odtworzenia błędów z innymi członkami zespołu. Możesz zrobić zrzut ekranu w emulatorze Androida, naciskając ikonę aparatu na pasku narzędzi po prawej stronie.

Możesz także użyć skrótu klawiaturowego Command + S, aby zrobić zrzut ekranu. Zrzut ekranu jest automatycznie zapisywany w folderze Pulpit.

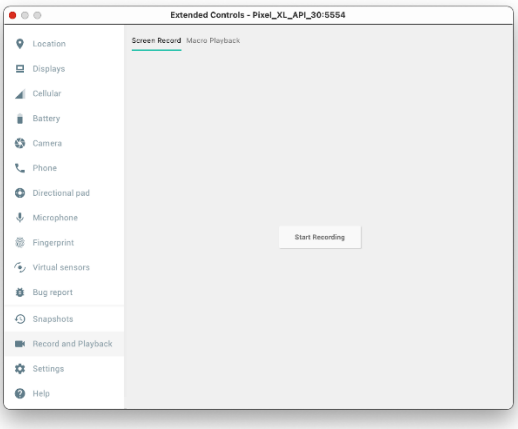
Nagraj działającą aplikację

Chociaż zrzuty ekranu mogą zawierać wiele informacji, czasami warto udostępnić nagrania uruchomionej aplikacji, aby pomóc innym odtworzyć coś, co spowodowało błąd. Emulator systemu Android oferuje kilka wbudowanych narzędzi, które ułatwiają przechwytywanie GIF (animowanego obrazu) uruchomionej aplikacji.

1. W narzędziach emulatora po prawej stronie kliknij przycisk Więcej (ostatnia opcja), aby wyświetlić dodatkowe opcje debugowania emulatora. Pojawi się okno z dodatkowymi narzędziami do symulacji funkcjonalności urządzeń fizycznych do celów testowych.



1. W menu po lewej stronie kliknij **Nagrywaj i odtwarzaj** , a zobaczysz ekran z przyciskiem, aby rozpocząć nagrywanie.



1. W tej chwili Twój projekt nie ma nic ciekawego do nagrania poza statycznym `TextView`. Zmodyfikujmy kod, aby aktualizować etykietę co kilka sekund, aby pokazać wynik dzielenia. W `division()`metodzie w `MainActivity`Dodaj wywołanie do `Thread.sleep(3)`przed wywołaniem do `Log()`. Metoda powinna teraz wyglądać następująco (zwróć uwagę, że pętla powinna powtórzyć się tylko 4 razy, aby uniknąć awarii).

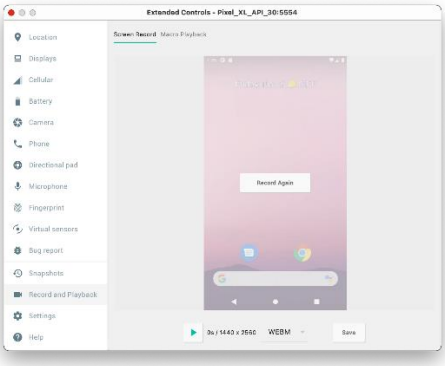
```
fun division() {
  val numerator = 60
  var denominator = 4
  repeat(4) {
    Thread.sleep(1)
    Log.v(TAG, "${numerator / denominator}")
    denominator--
  }
}
```

1. W `activity_main.xml`ustaw `id`wartość . `TextView_division_textview`

1. Wróć do `MainActivity.kt`programu , zastąp `Log.d()`następujące wywołanie funkcji `findViewById()`i , `setText()`aby ustawić tekst na iloraz.

```
findViewById<TextView>(R.id.division_textview).setText("${numerator / denominator}")
```

1. Uruchom swoją aplikację, a następnie natychmiast przełącz się na emulator. Po uruchomieniu aplikacji kliknij przycisk **Rozpocznij nagrywanie** w oknie Rozszerzone sterowanie. Powinieneś zobaczyć iloraz aktualizowany co trzy sekundy. Po kilkukrotnym zaktualizowaniu kliknij **Zatrzymaj nagrywanie** .



1. Domyślnie dane wyjściowe są zapisywane w .webmformacie. Użyj menu rozwijanego, aby wyeksportować dane wyjściowe jako plik GIF.

7. Gratulacje

Gratulacje! Na tej ścieżce nauczyłeś się, że:

- Debugowanie to proces rozwiązywania problemów w kodzie.
- Dziennik umożliwia drukowanie tekstu z różnymi poziomami dziennika i znacznikami.
- Ślad stosu zawiera informacje o wyjątku, takie jak dokładna funkcja, która go spowodowała, oraz numer wiersza, w którym wystąpił wyjątek.
- Podczas debugowania istnieje prawdopodobieństwo, że ktoś napotkał ten sam lub podobny problem i możesz użyć witryn takich jak StackOverflow, aby zbadać błąd.
- Możesz łatwo eksportować zarówno zrzuty ekranu, jak i animowane pliki GIF za pomocą emulatora Androida.

Ucz się więcej

- Logcat
- Debuguj swoją aplikację
- Dziennik
- Modyfikatory widoczności
- Korzystanie z emulatora Androida

Projekt: Aplikacja lemoniada

1. Zanim zaczniesz

To laboratorium programowania przedstawia nową aplikację o nazwie Lemonade, którą zbudujesz samodzielnie. To laboratorium kodowania przeprowadzi Cię przez kolejne kroki, aby ukończyć projekt, w tym konfigurację 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;
- Wracając do poprzedniej zawartości w Android Basics Unit 1, aby odświeżyć to, czego się nauczyłeś;
- Porównywanie kodu, o którym wiesz, że działa (tj. kodu podanego w projekcie lub wcześniejszego kodu rozwiązania z innych aplikacji w jednostce 1) z kodem, który piszesz.

Na początku może się to wydawać zniechęcające, ale jesteśmy w 100 procentach pewni, że jeśli udało ci się ukończyć część 1 , 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 1 kursu Android Basics in Kotlin.

Co zbudujesz

- Zbudujesz prostą aplikację Lemonade, korzystając z umiejętności, których nauczyłeś się w części 1.

Co będziesz potrzebował

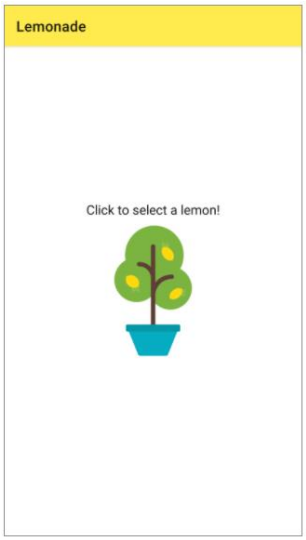
- Komputer z zainstalowanym Android Studio.

2. Przegląd aplikacji

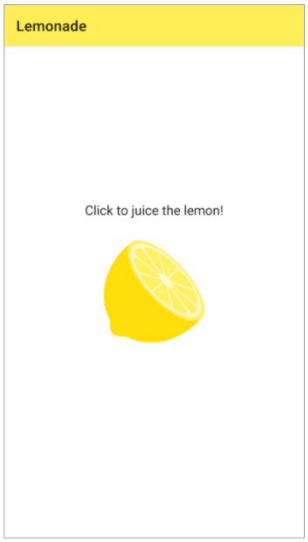
Witamy w aplikacji Project: Lemoniada!

Zrekrutowaliśmy Cię do naszego zespołu, aby pomóc nam urzeczywistnić naszą wizję tworzenia cyfrowej lemoniady. Celem jest stworzenie prostej, interaktywnej aplikacji mobilnej, która pozwoli ci wyciskać sok z cytryny aż do wypicia szklanki lemoniady. Potraktuj to jako metaforę, a może po prostu zabawny sposób na zabicie czasu!

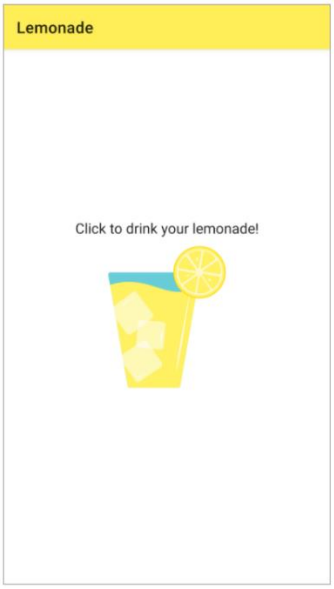
Gotowa aplikacja Lemonade będzie składać się z jednego ekranu. Gdy użytkownicy po raz pierwszy uruchamiają aplikację, są witani monitem o wybranie cytryny poprzez dotknięcie zdjęcia drzewa cytrynowego.



Stuknięcie w drzewo cytrynowe przedstawia użytkownikowi cytrynę, którą może stuknąć, aby „wycisnąć” nieokreśloną liczbę razy (dokładna liczba wymaganych przyciśnięć jest generowana losowo) przed przejściem do następnego ekranu.



Gdy użytkownik naciśnie odpowiednią liczbę razy, aby wycisnąć cytrynę, zobaczy obraz szklanki do „wypicia” lemoniady.



Po kliknięciu, aby wypić lemoniadę, szklanka wydaje się pusta, a użytkownik może ponownie dotknąć obrazu, aby powrócić do pierwszego ekranu i wybrać kolejną cytrynę z drzewa.

Aplikacja opiera się na prostocie i jest zorganizowana w ramach jednej czynności. Różne stany aplikacji (czy użytkownik wybiera cytrynę, ściska cytrynę, pije lemoniadę i wreszcie pustą szklankę) są reprezentowane przez coś, co nazywa się *maszyną stanów* . Brzmi to jak wymyślny termin teoretyczny, ale oznacza to, że stan aplikacji (tj. tekst i obraz wyświetlany użytkownikowi) jest określany przez zmienną zawierającą stan aplikacji (`select`, `squeeze`itd.). Stan aplikacji jest aktualizowany wraz z innymi potrzebnymi zmiennymi, a następnie konfigurowany jest interfejs użytkownika (ustawiając obraz i tekst) osobno po dokonaniu wszystkich aktualizacji.

Wszystkie zmienne dla stanu aplikacji zostały dla Ciebie zdefiniowane. Twoim zadaniem jest zbudowanie układu aplikacji i zaimplementowanie logiki, tak aby interfejs użytkownika przechodził między każdym stanem zgodnie z oczekiwaniami.

Testowanie kodu

W przypadku aplikacji Lemonade (i przyszłych projektów) otrzymasz kilka automatycznych testów, których możesz użyć do sprawdzenia, czy Twój kod działa zgodnie z oczekiwaniami.

Czym dokładnie są testy automatyczne? W tworzeniu oprogramowania można myśleć o „teście” jako o kodzie, który weryfikuje, czy inny kod działa. Odbywa się to poprzez sprawdzenie danych wyjściowych (takich jak zawartość elementów interfejsu użytkownika na ekranie), aby sprawdzić, czy mają one sens na podstawie danych wejściowych, znanych jako „przypadki testowe”. Projekt startowy dla aplikacji Lemonade zawiera kilka testów, które możesz uruchomić, aby upewnić się, że logikę zaimplementowano poprawnie. Testy omówimy bardziej szczegółowo później. Na razie czas pobrać kod startowy i zacząć budować aplikację Lemonade.

3. Rozpocznij

Pobierz kod projektu

Zauważ, że nazwa folderu to `android-basics-kotlin-lemonade-app`. Wybierz ten folder podczas otwierania projektu w Android Studio.

Adres URL kodu startowego:

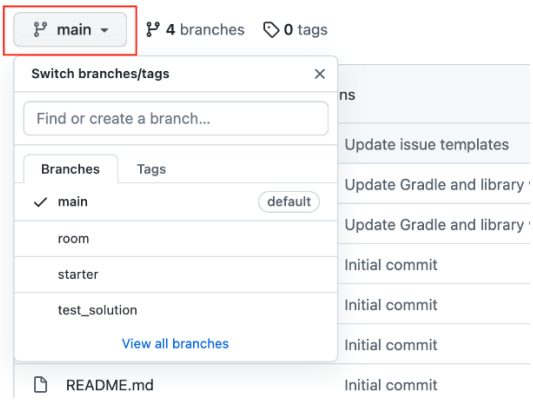
https://github.com/google-developer-training/android-basics-kotlin-lemonade-app

Nazwa oddziału z kodem startowym:main

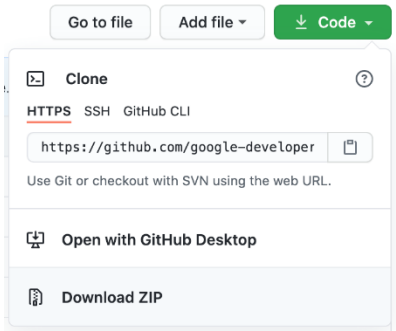
Aby uzyskać kod tego ćwiczenia z programowania i otworzyć go w Android Studio, wykonaj następujące czynności.

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** .



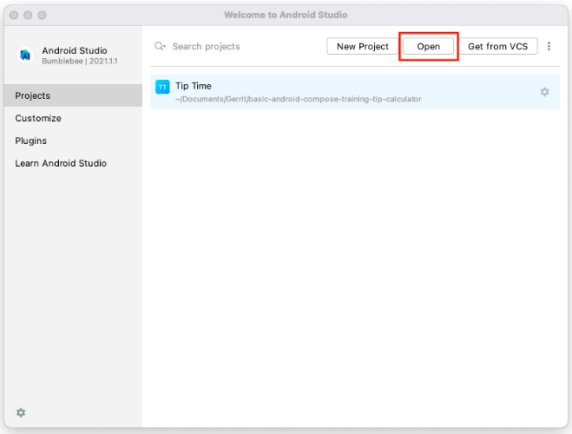
1. Na stronie GitHub projektu kliknij przycisk **Kod** , który wyświetli wyskakujące okienko.



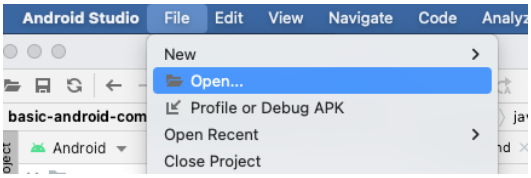
1. W wyskakującym okienku kliknij przycisk **Pobierz ZIP** , aby zapisać projekt na komputerze. Poczekaj na zakończenie pobierania.
2. Znajdź plik na swoim komputerze (prawdopodobnie w folderze **Pobrane).**
3. Kliknij dwukrotnie plik ZIP, aby go rozpakować. Spowoduje to utworzenie nowego folderu zawierającego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

Otwórz projekt w Android Studio

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

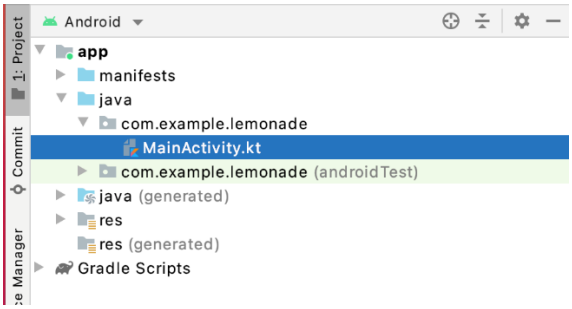


Uwaga: jeśli Android Studio jest już otwarte, wybierz opcję menu **Plik** > **Otwórz** .



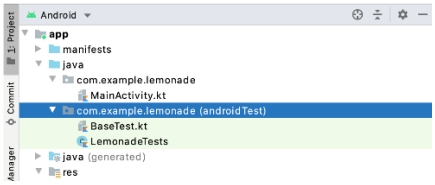
1. W przeglądarce plików przejdź do miejsca, w którym znajduje się rozpakowany folder projektu (prawdopodobnie w folderze **Pobrane**).
2. Kliknij dwukrotnie ten folder projektu.
3. Poczekaj, aż Android Studio otworzy projekt.
4. Kliknij przycisk **Uruchom** , aby skompilować i uruchomić aplikację. Upewnij się, że kompiluje się zgodnie z oczekiwaniami.

Poświęć chwilę na zapoznanie się z projektem startowym. Zwróć szczególną uwagę na `MainActivity.kt`plik.



W programie `MainActivity.kt`znajdziesz kilka zmiennych używanych do reprezentowania aktualnego stanu aplikacji. Użyjesz ich na późniejszym etapie, aby uczynić aplikację interaktywną. Chociaż ilość kodu tutaj może wydawać się przytłaczająca, nie musisz modyfikować żadnego kodu, który nie jest oznaczony jako TODO. Szczegółowe instrukcje znajdują się na kolejnych stronach.

Zauważysz również, że projekt zawiera również inny pakiet, **com.example.lemonade (androidTest)** .



Obejmuje to testy automatyczne, których będziesz używać do sprawdzania poprawności implementacji funkcji `MainActivity.kt`. Znowu o tym później. Na razie możesz zacząć tworzyć swoją aplikację, zaczynając od interfejsu użytkownika.

4. Zbuduj swój interfejs użytkownika

Aplikacja Lemonade wymaga tylko podstawowego układu; potrzebujesz tylko dwóch widoków, aby zaimplementować całą jego funkcjonalność.

1. A `TextView`, który zawiera instrukcje dla użytkownika.
2. Wyświetlająca grafikę `ImageView`opartą na aktualnym stanie aplikacji (np. cytryna do wyciskania).

Zbudujesz ten układ w `activity_main.xml`.

Korzystając ze swojej wiedzy na temat edytora układu, Twoim celem jest zbudowanie układu, który wygląda podobnie do poniższego, z widokami wyśrodkowanymi na ekranie i widokami `TextView`nad `ImageView`.

Wskazówka: używałeś już a `ConstraintLayout`podczas pracy z edytorem układu Android Studio. Układy wiązań dobrze sprawdzają się przy tworzeniu układów widoków piętrowych, takich jak w aplikacji Lemonade. Jeśli potrzebujesz odświeżenia na `ConstraintLayout`, wróć do ćwiczenia z kodowania aplikacji Tworzenie kartki urodzinowej.

5. Spraw, aby Twoja aplikacja była interaktywna

Gdy układ będzie gotowy, otwórz `MainActivity.kt`. Tutaj zaimplementujesz całą logikę aplikacji. Zauważysz, że jest już sporo kodu. Zaznaczono również wiele komentarzy `// TODO:`(przykład poniżej). **To są zadania do wykonania** .

```
/**
 * Set up the view elements according to the state.
 */
private fun setViewElements() {
    val textAction: TextView = findViewById(R.id.text_action)
    // TODO: set up a conditional that tracks the lemonadeState

    // TODO: for each state, the textAction TextView should be set to the corresponding string from
    //  the string resources file. The strings are named to match the state

    // TODO: Additionally, for each state, the lemonImage should be set to the corresponding
    //  drawable from the drawable resources. The drawables have the same names as the strings
    //  but remember that they are drawables, not strings.
}
```

Istnieją trzy podstawowe rzeczy, które musisz wdrożyć, aby aplikacja lemoniady działała.

1. Skonfiguruj tak, `lemonImage` `ImageView`aby odpowiadał na dane wprowadzane przez użytkownika.
2. Zaimplementuj `clickLemonImage()`, aby zaktualizować stan aplikacji.
3. Zaimplementuj `setViewElements()`, aby zaktualizować interfejs użytkownika na podstawie bieżącego stanu aplikacji.

Przyjrzyjmy się każdemu zadaniu pojedynczo.

Krok 1: Skonfiguruj ImageView

Dotknięcie widoku obrazu powinno przenieść aplikację z jednego stanu do drugiego. Pod koniec `onCreate()`zauważ, że należy ustawić dwa odbiorniki.

1. `setOnClickListener()`powinien zaktualizować stan aplikacji. Metodą na to jest `clickLemonImage()`.
2. `setOnLongClickListener()`reaguje na zdarzenia, w których użytkownik długo naciska obraz (np. użytkownik stuka w obraz i nie zwalnia natychmiast palca). W przypadku długich wydarzeń prasowych na dole ekranu pojawia się widżet zwany paskiem przekąsek, informujący użytkownika, ile razy wycisnął cytrynę. Odbywa się to za pomocą `showSnackbar()`metody.

Uwaga: ważne jest, abyś NIE zmieniał nazwy żadnej z istniejących metod. W przeciwnym razie testy mogą nie zostać uruchomione.

W następnym kroku zaimplementujesz logikę zmiany stanu aplikacji.

Krok 2: Zaimplementuj clickLemonImage()

Po wykonaniu poprzedniego kroku `clickLemonImage()`metoda jest teraz wywoływana za każdym razem, gdy użytkownik dotknie obrazu. Ta metoda jest odpowiedzialna za przenoszenie aplikacji z obecnego stanu do następnego i aktualizowanie wszelkich zmiennych w razie potrzeby. Istnieją cztery możliwe stany: `SELECT`, `SQUEEZE`, `DRINK`i, `RESTART`; aktualny stan jest reprezentowany przez `lemonadeState`zmienną. Ta metoda musi zrobić coś innego dla każdego stanu.

1. `SELECT`: Przejście do `SQUEEZE`stanu, ustaw `lemonSize`(ilość potrzebnych ściśnięć) wywołując `pick()`metodę i ustawiając `squeezeCount`(ilość wyciśnięć cytryny przez użytkownika) na 0.
2. `SQUEEZE`: Zwiększ `squeezeCount`o 1 i zmniejsz `lemonSize`o 1. Pamiętaj, że cytryna będzie wymagała różnej liczby przyciśnięć, zanim aplikacja będzie mogła zmienić swój stan. Przejście do `DRINK`stanu tylko wtedy, gdy nowy `lemonSize`jest równy 0. W przeciwnym razie aplikacja powinna pozostać w `SQUEEZE`stanie.
3. `DRINK`: Przejście do `RESTART`stanu i ustaw na `lemonSize`-1.
4. `RESTART`: Powrót do `SELECT`stanu.

Po obsłużeniu wszystkich aktualizacji i przejść między stanami pamiętaj, aby wywołać `setViewElements()`aktualizację interfejsu użytkownika na podstawie nowego stanu.

Wskazówka: instrukcja when to świetny sposób na sprawdzenie wielu możliwych warunków pojedynczej wartości, takich jak `lemonadeState`.

Krok 3: Zaimplementuj setViewElements()

Metoda `setViewElements()`odpowiada za aktualizację interfejsu użytkownika na podstawie stanu aplikacji. Tekst i obraz należy zaktualizować o wartości pokazane poniżej, aby pasowały do `lemonadeState`.

`SELECT`:

- Tekst: Kliknij, aby wybrać cytrynę!
- Obraz:`R.drawable.lemon_tree`

SQUEEZE:

- Tekst: Kliknij, aby wycisnąć sok z cytryny!
- Obraz:R.drawable.lemon_squeeze

DRINK:

- Tekst: Kliknij, aby wypić lemoniadę!
- Obraz:R.drawable.lemon_drink

RESTART:

- Tekst: Kliknij, aby zacząć od nowa!
- Obraz:R.drawable.lemon_restart

Wskazówka: kod startowy ma już zmienną dla lemonImagepodczas ustawiania detektorów w onCreate().

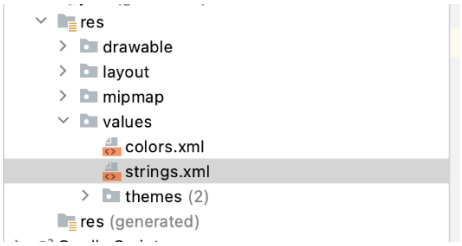
Dodatkowy kredyt: Zamiast zakodować ciągi na stałe w kodzie Kotlin, rozważ utworzenie zasobów ciągów tych wartości w strings.xml.

Jak korzystać z zasobów ciągów

W Androidzie prawie wszystko jest zasobem. Definiowanie zasobów, do których możesz następnie uzyskać dostęp w swojej aplikacji, jest istotną częścią rozwoju Androida.

Zasoby są używane do wszystkiego, od definiowania kolorów, obrazów, układów, menu i wartości ciągów. Zaletą tego jest to, że nic nie jest zakodowane na sztywno. Wszystko jest zdefiniowane w tych plikach zasobów, a następnie można się do nich odwoływać w kodzie aplikacji. Najprostszym z tych zasobów (i najbardziej powszechnym) jest użycie zasobów tekstowych, aby umożliwić elastyczny, zlokalizowany tekst.

Ciągi znaków lub tekst statyczny można przechowywać w oddzielnym pliku o nazwie strings.xml w podfolderze wartości folderu res.



Dla każdego fragmentu tekstu, który chcesz wyświetlić w swojej aplikacji (tj. etykiety przycisku lub tekstu wewnątrz TextView), powinieneś najpierw zdefiniować tekst w res/values/strings.xmlpliku. Każdy wpis jest kluczem (reprezentującym identyfikator tekstu) i wartością (sam tekst). Na przykład, jeśli chcesz, aby przycisk wyświetlał „ **Prześlij** ”, dodaj następujący zasób ciągu do res/values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Hello!</string>
  <string name="submit_label">Submit</string>
```

```
</resources>
```

Aby uzyskać dostęp do zasobu bezpośrednio w kodzie, po prostu użyj metody `getResources.getString()`lub w `getString()`celu uzyskania dostępu do wartości podanej w identyfikatorze zasobu, `R.string.submit_label`.:

```
val submitText = getResources().getString(R.string.submit_label)
```

Aby bezpośrednio ustawić tekst z zasobu ciągu do TextView, można wywołać `setText()`obiekt TextView i przekazać identyfikator zasobu.

```
val infoTextView: TextView = findViewById(R.id.info_textview)

infoTextView.setText(R.string.info_text)
```

Zasoby tekstowe mogą również zawierać znaki specjalne do formatowania tekstu. Na przykład możesz mieć zasób ciągu, który umożliwia wstawienie innego fragmentu tekstu do ciągu.

```
<string name="ingredient_tablespoon">%1$d tbsp of %2$s</string>
```

W kodzie można uzyskać dostęp do zasobu ciągu i sformatować go, przekazując argumenty.

```
getResources().getString(R.string.ingredient_tablespoon, 2, "cocoa powder")
```

Podczas deklarowania zasobu typu string każdy argument jest numerowany w kolejności, w jakiej się pojawia (`1`, `2`itd.) i ma literę identyfikującą typ (`d`dla liczby dziesiętnej, `s`dla łańcucha itp.). Argumenty prawidłowego typu mogą zostać przekazane do wywołania `getString()`.

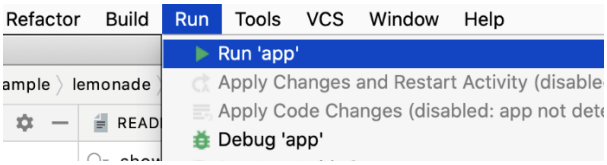
```
2 tbsp of cocoa powder
```

Aby dowiedzieć się więcej, zapoznaj się z dokumentacją dotyczącą zasobów ciągów .

Uwaga: możesz zobaczyć przykład zasobów ciągów przy użyciu argumentów w aplikacji Lemonade z `squeeze_count`zasobem, do którego dostęp uzyskuje się w `showSnackbar()`metodzie w `MainActivity`.

6. Uruchom swoją aplikację

Po zbudowaniu interfejsu użytkownika aplikacji i zaimplementowaniu głównej aktywności nadszedł czas, aby zobaczyć swoją ciężką pracę w akcji. Uruchom aplikację za pomocą menu **Uruchom > Uruchom aplikację,** a uruchomi się emulator.



Aplikacja powinna być teraz w pełni interaktywna i powinieneś być w stanie dotknąć widoku obrazu, aby przejść między stanami.

Gdy cytryna jest wyświetlana na ekranie, możesz również spróbować nacisnąć i przytrzymać (naciśnij i przytrzymaj), ImageViewaby zobaczyć pasek z przekąskami na dole, pokazujący całkowitą liczbę wyciśnięć cytryny. Poświęć trochę czasu, aby kilka razy przejrzeć aplikację przez wszystkie stany. Poświęć chwilę, aby pogratulować sobie ciężkiej pracy!

Wskazówka: jeśli nie masz żadnych urządzeń wirtualnych, możesz je utworzyć, postępując zgodnie z instrukcjami podanymi w sekcji Tworzenie i uruchamianie pierwszej aplikacji na Androida .

7. Instrukcje testowania

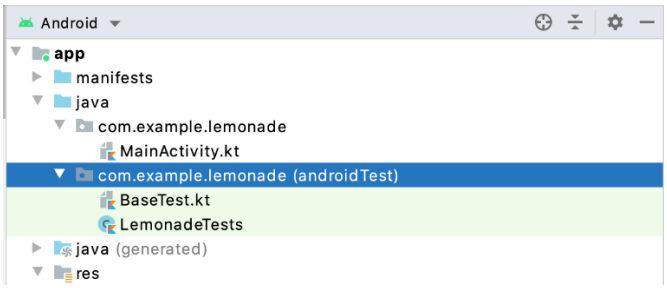
Przetestuj swoją aplikację

Zakończyłeś wdrażanie aplikacji Lemonade, ale w profesjonalnym tworzeniu oprogramowania samo pisanie kodu rzadko jest ostatnim krokiem. Oprócz kodu aplikacji aplikacje o profesjonalnej jakości zawierają również kod testowy, który jest uruchamiany w celu sprawdzenia, czy kod działa zgodnie z oczekiwaniami i czy zmiany w kodzie nie wprowadzają żadnych nowych błędów, proces zwany testowaniem automatycznym. Chociaż nauczanie automatycznego testowania wykracza poza zakres tego projektu, aplikacja Lemonade jest dołączona do niektórych testów, które pomogą Ci zweryfikować, czy projekt został prawidłowo zaimplementowany. Możesz użyć tego jako formy samooceny, aby sprawdzić, czy spełniłeś wszystkie wymagania projektu i czy potrzebne są jakiekolwiek zmiany w Twojej aplikacji.

Czym dokładnie jest „test”? Testy to po prostu fragmenty kodu zawarte w projekcie Android Studio, które uruchamiają część kodu aplikacji i mogą być „zaliczone” lub „niepowodzenie” w zależności od tego, czy kod aplikacji zachowuje się zgodnie z oczekiwaniami.

Wskazówka: Zazwyczaj komunikaty o niepowodzeniu testu będą specyficzne dla funkcji testowej. W poniższym przykładzie test zakończony niepowodzeniem sprawdza zawartość widoku tekstu. Jeśli jednak nie masz pewności, co oznacza komunikat o niepowodzeniu testu, zawsze możesz wkleić część błędu do Google. Są szanse, że ktoś wcześniej zadał to samo pytanie!

Gdzie więc znajdujesz i uruchamiasz testy swojej aplikacji? Testy aplikacji Lemonade znajdują się w *celu* testowania . Cel jest po prostu terminem programistycznym dla zbioru klas, które są ze sobą powiązane **.** Na przykład aplikacja Lemonade istnieje w miejscu docelowym o nazwie „app", podczas gdy testy znajdują się w miejscu docelowym o nazwie „LemonadeTests". Chociaż obiekt docelowy LemonadeTests może uzyskać dostęp do kodu z obiektu docelowego aplikacji, są one całkowicie oddzielne, a kod aplikacji nie zawiera żadnego kodu testowego.



Podczas przeglądania plików w widoku „Android", cel testu pojawi się z taką samą nazwą pakietu jak aplikacja, ale z (androidTest) w nawiasach.

Istnieje również kilka kluczowych terminów, które należy znać, odnosząc się do kodu testowego.

- **Test Suite** — cel, który zawiera wszystkie *przypadki testowe* .
- **Test Case** - klasa składająca się z pojedynczych testów dla powiązanej funkcjonalności (aplikacja Lemonade miała tylko jeden przypadek testowy, ale większe aplikacje często mają ich znacznie więcej).
- **Test** — funkcja, która testuje jedną konkretną rzecz.

Przypadek testowy może mieć wiele testów, a zestaw testów projektu może mieć wiele przypadków testowych.

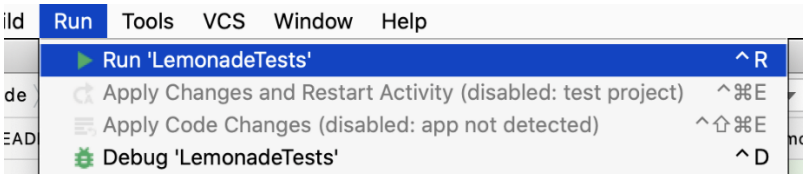
Przeprowadzanie testów

Aby uruchomić testy, możesz wykonać jedną z poniższych czynności.

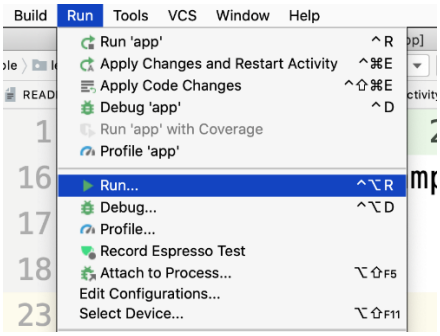
W przypadku pojedynczego przypadku testowego otwórz klasę przypadku testowego i kliknij zieloną strzałkę po lewej stronie deklaracji klasy. Następnie możesz wybrać z menu opcję Uruchom. Spowoduje to uruchomienie wszystkich testów w przypadku testowym.

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** .

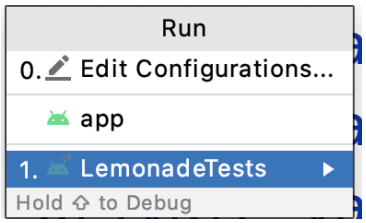
Jeśli masz wiele przypadków testowych, możesz również uruchomić cały zestaw testów. Podobnie jak w przypadku uruchamiania aplikacji, tę opcję znajdziesz w menu **Uruchom** .



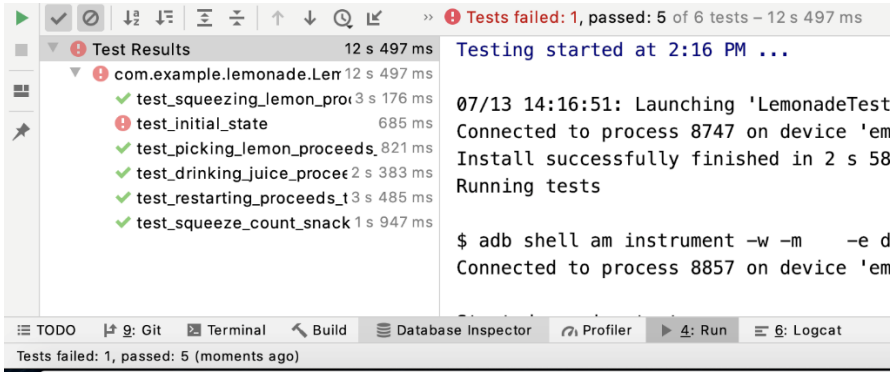
Zwróć uwagę, że Android Studio domyślnie dopasuje się do ostatniego uruchomionego celu (aplikacji, celów testowych itp.), więc jeśli w menu nadal pojawi się komunikat **Uruchom > Uruchom „aplikację”** , możesz uruchomić cel testu, wybierając **Uruchom > Uruchom** .



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 a `TextView`używa określonego zasobu ciągu. Jednak test kończy się niepowodzeniem. Tekst po „Oczekiwany" i „Oczekiwany" nie jest zgodny, co oznacza, że wartość oczekiwana przez test nie jest zgodna z wartością z uruchomionej aplikacji. W tym przykładzie ciąg znaków użyty w `TextView`nie jest in-fact `squeeze_count`, zgodnie z oczekiwaniami testu.

8. Opcjonalnie: udostępnij swoją aplikację!

Po wypiciu wielu kieliszków lemoniady zrób zrzut ekranu swojego ulubionego ekranu i udostępnij go na Twitterze, aby pokazać, czego się nauczyłeś. Oznacz @AndroidDev i dodaj hashtag **#AndroidBasics** .