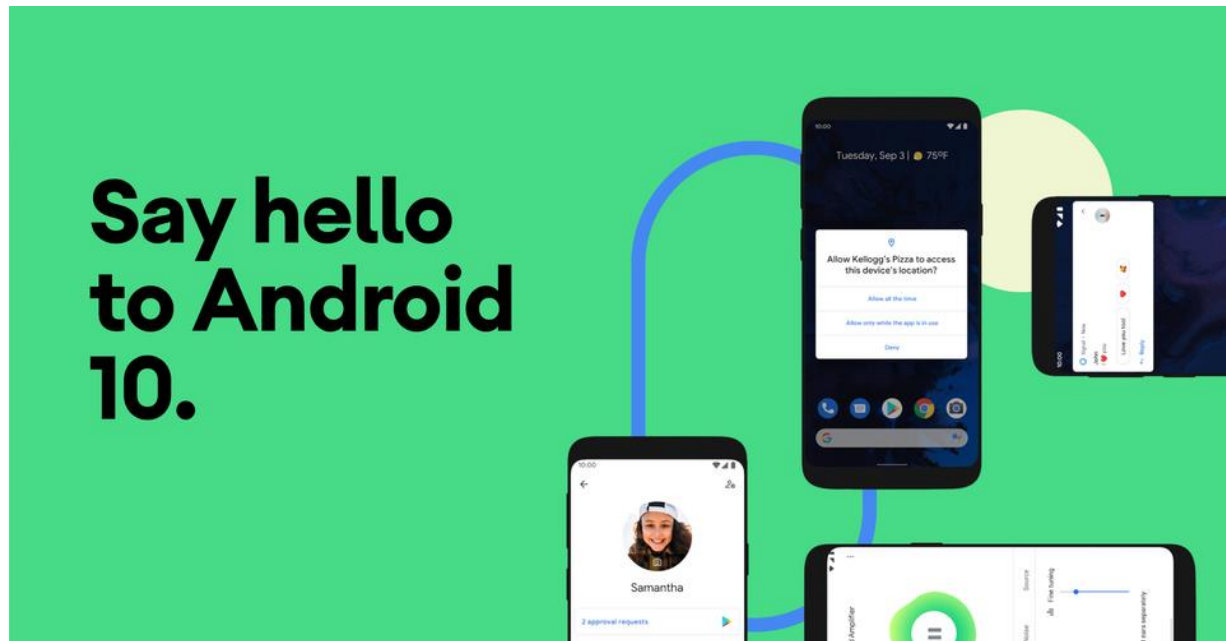


Aplikacje mobilne. Projektowanie aplikacji Android. Kształt i specyfika aplikacji dla Androida #1

Android – mobilny system operacyjny. Wprowadzenie i architektura.

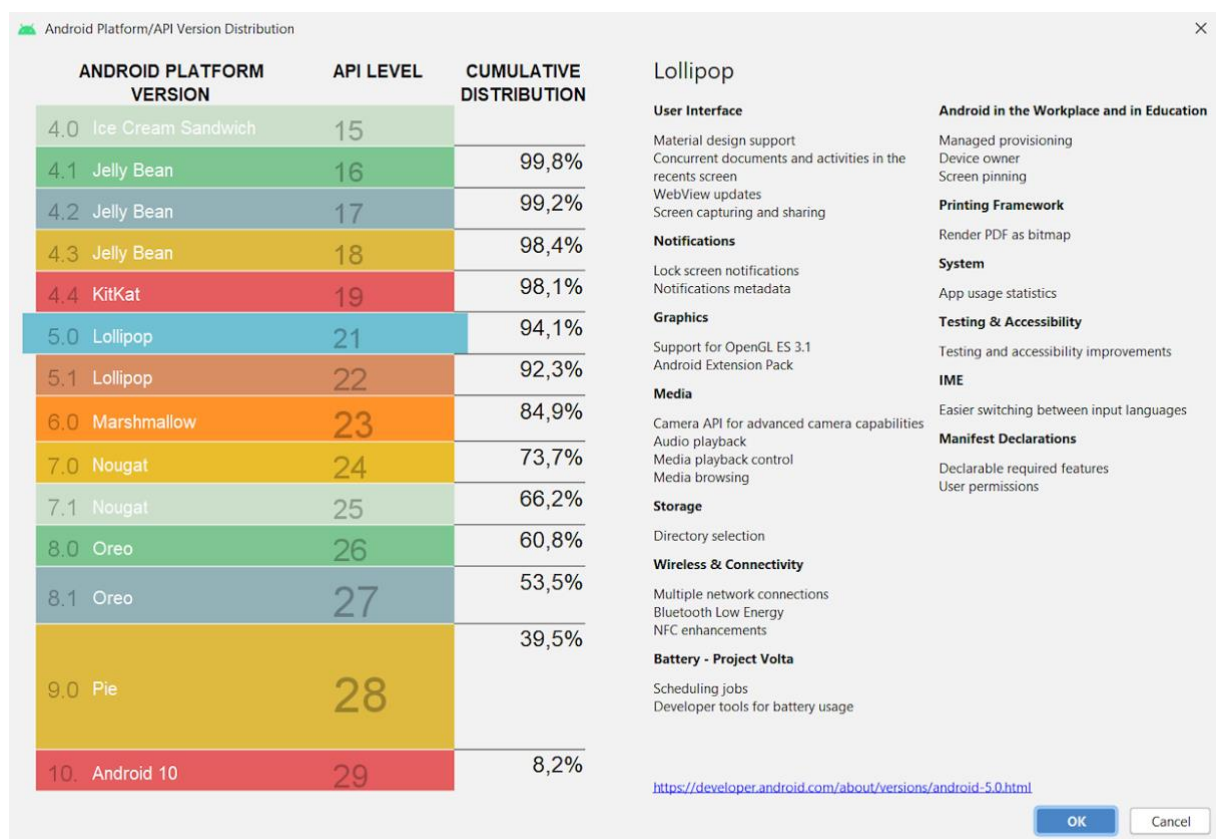
Android to przede wszystkim system operacyjny z jądrem Linux dla urządzeń mobilnych takich jak telefony komórkowe, smartfony, tablety (tablety PC) i netbooki czy też smart tv, dekodery i tunery satelitarne. Jedną z zalet tworzenia aplikacji systemu Android jest otwartość tej platformy. System ten jest darmowym oprogramowaniem na licencji open source. Dzięki temu każdy może przeanalizować kod źródłowy tego systemu i zobaczyć, jak zaimplementowane są pewne jego funkcje. **Obecna wersja to Android 10.** W dziesiątej generacji Androida postawiono na lepszą interakcję smartfona z użytkownikiem, rozpowszechnianie ciemnego motywu oraz kilka innych usprawnień.



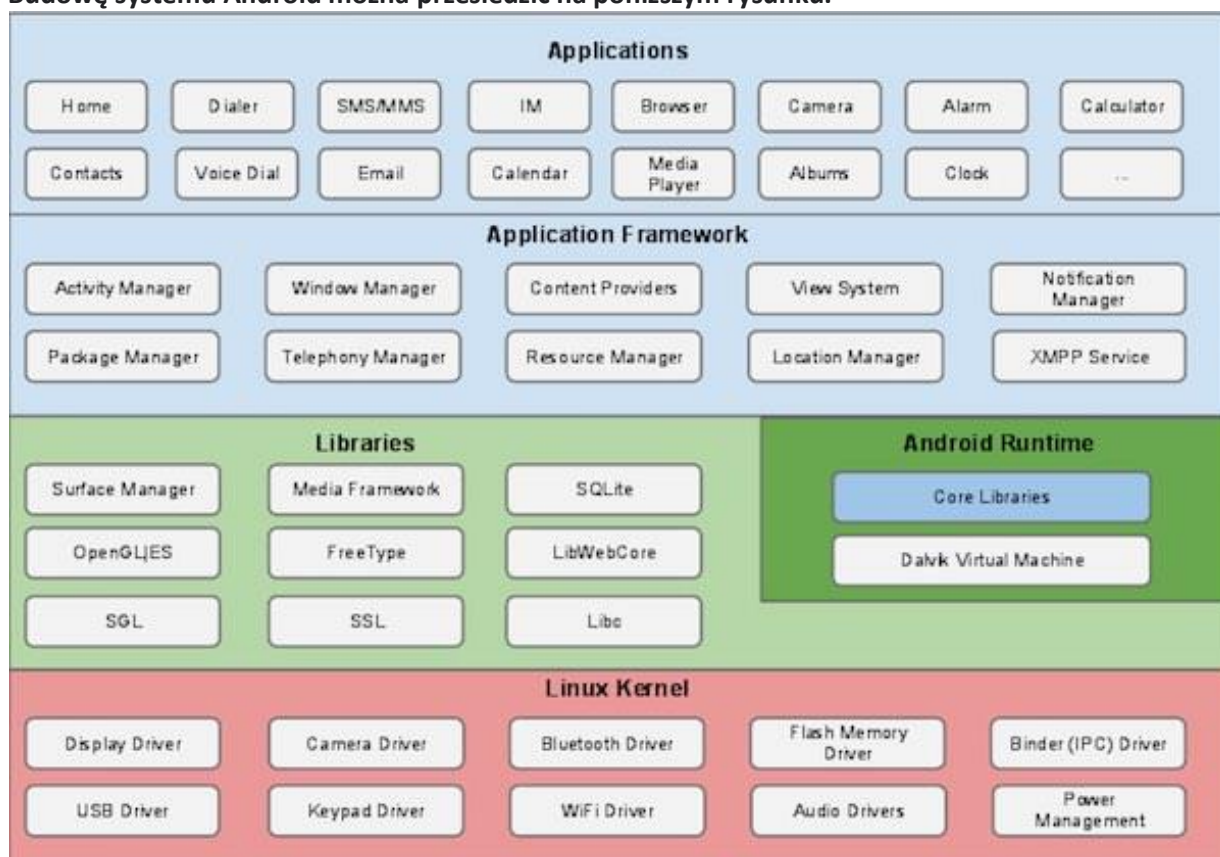
Z kolei zapowiadany system Android 11 pojawi się na rynku w drugiej połowie 2020 roku.

Więcej o Android 10 : <https://www.android.com/android-10/>

Obecna fragmentacja systemu Android : Informacje o dystrybucji są udostępniane z takim samym zestawem danych jak wcześniej (numer wersji, API i udział), choć podobno Google pracuje nad udostępnieniem większej ilości danych programistom w ciągu najbliższych kilku miesięcy. Google chwali się, że Android 10 w ciągu pięciu miesięcy od oficjalnej premiery kontrolował pracę 100 mln urządzeń, czyli o 28% większą liczbę niż Android Pie po upływie analogicznego okresu. Aktualnie, po 10 miesiącach od publicznego udostępnienia, Android 10 jest zainstalowany już na ponad 400 mln sprzętów.



Budowę systemu Android można prześledzić na poniższym rysunku.



Prześledźmy najważniejsze elementy architektury systemu.

- Linux Kernel (jądro linuxowe) - Android opiera się na wersji jądra 2.6 dla podstawowych usług systemowych, takich jak bezpieczeństwo, zarządzanie pamięcią, zarządzanie procesami, stos sieciowy i model sterownika. Jądro działa również jako warstwa abstrakcji pomiędzy sprzętem i resztą stosu oprogramowania.
- Android Runtime (środowisko uruchomieniowe) - Android zawiera zbiór bibliotek, które dostarczają większość funkcji dostępnych w bibliotekach podstawowych języka Java. Każda aplikacja działa we własnym procesie, z własnej instancji maszyny wirtualnej Dalvik. Dalvik została napisana tak, że na urządzeniu można uruchomić wiele maszyn wirtualnych. Dalvik VM wykonuje pliki wykonywalne (.dex) skonstruowane tak, aby zużywały minimalną ilość pamięci
- Libraries (biblioteki) - Android zawiera zestaw bibliotek C / C++ używanych przez różne elementy systemu. Możliwości te są udostępnione programistom poprzez Application Framework.
- Application Framework (framework aplikacji) - Deweloperzy mają pełny dostęp do tego samego API, używanego przez aplikacje podstawowe systemu. Architektury aplikacji ma na celu uproszczenie ponownego używania komponentów, każda aplikacja może publikować swój interfejs i każda inna aplikacja może wówczas z niego skorzystać (z zastrzeżeniem ograniczeń bezpieczeństwa). Ten sam mechanizm pozwala na wymianę komponentów przez użytkownika.
- Framework oferuje zestaw usług i systemów, w tym:
 Bogaty i elastyczny zestaw widoków (Views), które można wykorzystać do budowania aplikacji, w tym list, siatek, pól tekstowych, przycisków, zagnieżdżenia przeglądarki internetowej (WebView)
 Dostawców treści (Content Providers), które pozwalają aplikacjom dostęp do danych z innych aplikacji, (np. takich jak Kontakty), lub dzielić się swoimi danymi
 Menedżer zasobów (Resource Manager), umożliwiając dostęp do zasobów, takich jak zlokalizowanych treści, grafik, plików układu aplikacji
 Notification Manager, który umożliwia wszystkim aplikacjom wyświetlanie powiadomień w pasku stanu.
 Activity Manager, który zarządza cyklem życia aplikacji.

Ważne!

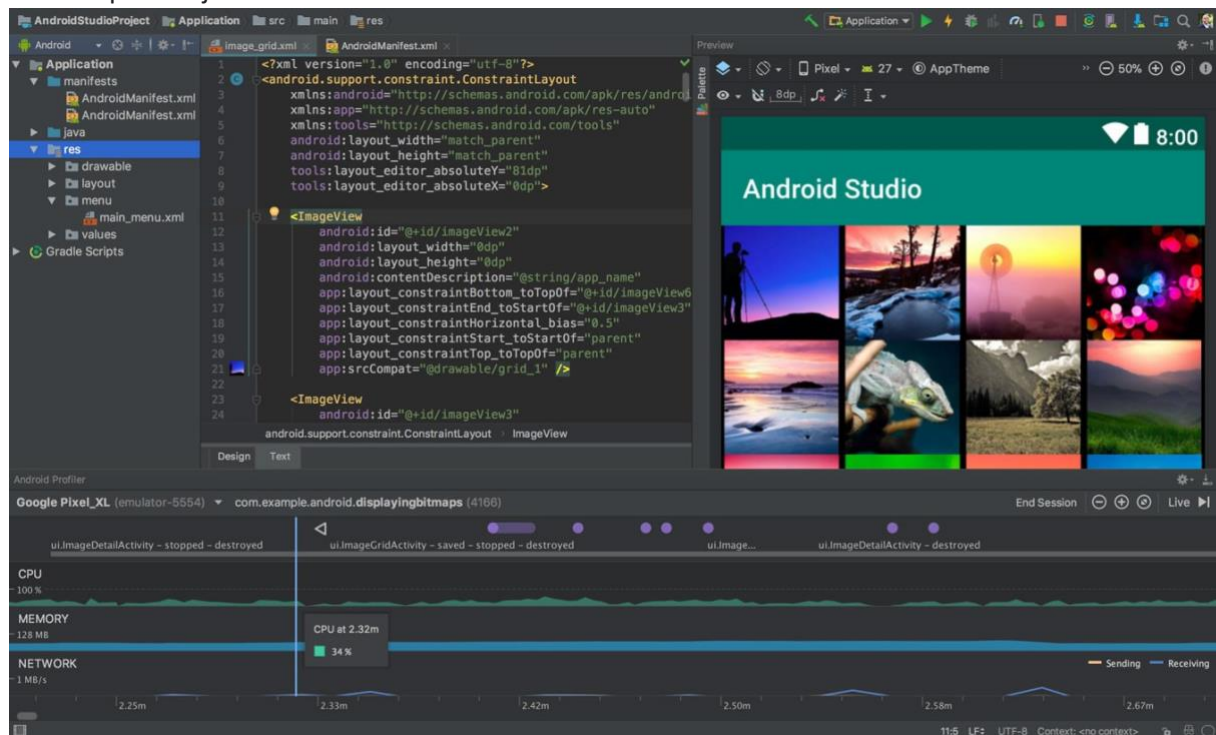
- Każda aplikacja działa w obrębie własnej maszyny Dalvik (jest odizolowana od innych).
- Aplikacje są kodem zarządzanym.
- Każda aplikacja działa jako osobny użytkownik systemu Linux, w osobnym procesie.
- Każda aplikacja musi uzyskać odpowiednie uprawnienia, aby skorzystać z „czułych” elementów systemu (kontakty, rozmowy, karta SD itp.) albo z danych udostępnianych przez inne aplikacje.
- Aplikacje muszą być odpowiednio podpisane przy użyciu certyfikatów, aby można je było publikować i instalować.

Programowanie aplikacji Android. Podstawy

W pierwszej kolejności wykonamy prostą aplikację na system Android za pomocą środowiska programistycznego **Android Studio**. Następnie uruchomimy ją w wirtualnym środowisku i na smartfonie. Potem utworzymy nowy interfejs dla aplikacji, który pobiera dane wejściowe od użytkownika i otwiera drugi ekran w aplikacji. **Kurs prowadzony będzie z wykorzystaniem nowoczesnego języka programowania Kotlin.**

Aplikacja Android Studio to środowisko programistyczne (IDE) stworzone przez Google na bazie IntelliJ, które kierowane jest do developerów aplikacji na Androida. Pozwala ono wygodnie projektować, tworzyć i debugować własne programy na najpopularniejszą obecnie platformę systemową dla urządzeń mobilnych. Decydując się na korzystanie z Android Studio użytkownik

otrzymuje środowisko programistyczne z przejrzystym i konfigurowalnym interfejsem graficznym, nie wspominając o funkcji kolorowania składni czy mechanizmie zakładki, pozwalającym na pracę z wieloma plikami jednocześnie.



W środowisku Android Studio z pewnością odnajdą się zarówno początkujący programiści, jak również ci bardziej zaawansowani. Twórcy postarali się, aby poza samym narzędziem do pisania kodu developerzy znaleźli w nim także inne, przydatne podczas codziennej pracy funkcje i narzędzia. W tym miejscu wspomnieć należy m.in. o debuggerze, liście ToDo, pozwalającej organizować zadania do wykonania w obrębie danego projektu czy mechanizmowi wirtualnych urządzeń do testowania tworzonych aplikacji.

Poza wspomnianymi funkcjami w oprogramowaniu znajdziemy również wiele innych, mniejszych, ale niemniej użytecznych funkcji jak chociażby możliwość testowania wyglądu i zachowania interfejsu aplikacji w różnych rozdzielczościach czy orientacjach ekranu bez konieczności kompilacji oraz uruchamiania całego projektu.

Środowisko Android Studio należy pobrać ze strony : <https://developer.android.com/studio>
Pomocnym źródłem jest też strona z informacjami, tutorialami oraz manualami : <https://developer.android.com>



Najważniejsze elementy aplikacji Android

Android pozwala wykorzystywać różne zasoby dla różnych urządzeń jak smartfony czy tablety np. jeśli potrzebujemy wykorzystać w naszej aplikacji aparat/kamerę to aplikacja na samym początku może sprawdzić czy urządzenie te posiada je, a następnie uruchomić je bądź nie. Idea funkcjonowania aplikacji mobilnej obejmuje cztery podstawowe elementy składowe

- Kontekst (obiekt typu Context)- Kontekst jest środowiskiem, w którym uruchamiana jest zaprojektowana aplikacja mobilna. Za pośrednictwem kontekstu istnieje możliwość zarządzania aplikacją i uzyskiwania dostępu do parametrów aplikacji oraz urządzenia, na którym została uruchomiona.
- Aktywność (obiekt typu Activity) - Aktywności stanowią podstawowy budulec do tworzenia zaplanowanej funkcjonalności aplikacji mobilnej i zawierają ich implementację. Oznacza to, że wywoływanie funkcji aplikacji mobilnej odbywa się przez uruchomienie Aktywności. W danym zachowaniu można wywoływać kolejne Aktywności, dzięki czemu uzyskujemy możliwość realizacji nawet bardzo złożonych scenariuszy. Podstawową właściwością Aktywności jest krótki czas jej wykonania oraz kontakt z użytkownikiem aplikacji lub innymi aplikacjami.
- Intencja (obiekt typu Intent) - Intencje stanowią mechanizm obsługi zdarzeń występujących w aplikacji mobilnej (komunikaty asynchroniczne). Z wykorzystaniem mechanizmu Intencji możliwe jest uruchomienie Aktywności (także w innej aplikacji) lub serwisu w momencie wystąpienia określonego zdarzenia. Mechanizm Intencji służy także do komunikacji między różnymi aplikacjami uruchomionymi na urządzeniu mobilnym oraz do obsługi zdarzeń pochodzących z innych aplikacji (np. połączenie przychodzące) i wyposażenia urządzenia (np. niski stan baterii, wykonano zdjęcie).
- Usługa/Serwis (obiekt typu Service) Usługi można traktować jako specyficzne zastosowanie Aktywności. Usługi zawierają implementację funkcji, które nie wykorzystują interfejsu komunikacji z użytkownikiem oraz wymagają długiego czasu działania. Może on wynikać z czasochłonnych algorytmów obliczeniowych lub z potrzeby dostępu do bazy danych (w tym zewnętrznych baz danych) i konieczności wysłania lub pobrania dużej ilości danych. Cechą charakterystyczną usługi jest fakt działania „w tle”. W postaci usługi często implementuje się funkcje działające cyklicznie.
- Dostawcy treści (obiekt typu ContentProvider). Dostawcy treści (dane) zapewniają dostęp do swoich zasobów innym aplikacjom zainstalowanym na urządzeniu. Udostępniane dane mogą być przechowywane przez dostawcę w bazie danych SQLite lub w innej trwałej postaci. Dostawcy treści mogą też być związani komponentami sprzętowymi zainstalowanymi w urządzeniu – takimi jak GPS.

Środowisko Android Studio korzysta z systemu **Gradle Build**, który dokonuje kompilacji kodu — tworzy plik APK (format ten umożliwia instalację aplikacji). System Gradle obsługuje również zależności pomiędzy elementami projektu — wykonuje np. operacje dodawania bibliotek, z których korzysta aplikacja.

Plik AndroidManifest.xml opisuje wszystkie składniki aplikacji na Androida i jest odczytywany przez system po uruchomieniu aplikacji. Aplikacja musi zadeklarować wszystkie jego składniki w tym pliku, które muszą znajdować się w katalogu głównym katalogu aplikacji. Manifest składa się z kilku elementów oprócz deklarowania składników aplikacji, takich jak:

- Identyfikuje uprawnienia użytkownika wymagane przez aplikację, takie jak dostęp do internetu czy dostęp do kont kontaktów użytkownika z odczytem.
- Deklaruje minimalny poziom API wymagany przez aplikację, na podstawie których używane są aplikacje API.

- Deklaruje funkcje sprzętowe i programowe używane lub wymagające aplikacji, takie jak aparat fotograficzny, usługi bluetooth lub ekran multitouch.
- Deklaruje bibliotekę API, z którą aplikacja musi być powiązana (poza interfejsami API systemu Android), na przykład w bibliotece Map Google .

Podstawy języka Kotlin

Kotlin to język programowania, który może działać na wirtualnej maszynie Java – JVM, który jest głównie rozwijany przez programistów JetBrains. Nazwa języka pochodzi od wyspy Kotlin niedaleko Petersburga. Kotlin jest zaprojektowany z myślą o pełnej interoperacyjności z językami działającymi na maszynie wirtualnej Javy czyli np. czystej Javy.

Google ogłosiło, że Kotlin jest jednym z oficjalnie obsługiwanych języków programowania w Android Studio, zaś społeczność Androida migruje w szybkim tempie z Javy do Kotlinu. Nauka programowania w Androidzie jest bardzo łatwa, biorąc pod uwagę właściwy kierunek i właściwe kroki.



Oprócz klas i metod (nazywanych przez dokumentację jako member functions), Kotlin wspiera także programowanie proceduralne za pomocą funkcji. Podobnie jak w Javie, klasycznym punktem wejścia do programu w Kotlinie jest funkcja main, do której przekazywana jest tablica z argumentami podanymi w linii poleceń.

Przykład programu Hello world napisanego w języku Kotlin:

1. Przykład zapisu funkcji/metody w języku Kotlin:

```
fun main() {  
    println("Hello World")  
}
```

2. Przykład zapisu klasy w języku Kotlin:

```
class Greeter(val name: String) {  
    fun greet() {  
        println("Hello, $name")  
    }  
}  
  
fun main(args: Array<String>) {  
    Greeter(args[0]).greet()  
}
```

W Kotlinie zmienne są podzielone na *val* i *var*. Te przypisywane jednorazowo, tylko do odczytu oznaczają się jako *val*, a te z możliwością zmiany wartości jako *var*.

Przykład:

```
var a: Int = 15  
val b = 15 // Int  
val c = "" // String  
val d: String // wartość do odczytu, przypisana później  
var e: Int = null // błąd - Int nie jest Nullable  
var f: Int? = null  
var g // błąd - brak typu lub wartości  
a = 10  
b = 10 // błąd - val jest tylko do odczytu  
d = ""  
d = "" // błąd - d zostało już przypisane
```

Typów podstawowych w języku Kotlin jest 8 i składają się one z Double, Float, Long, Int, Short, Byte, Char i Boolean.

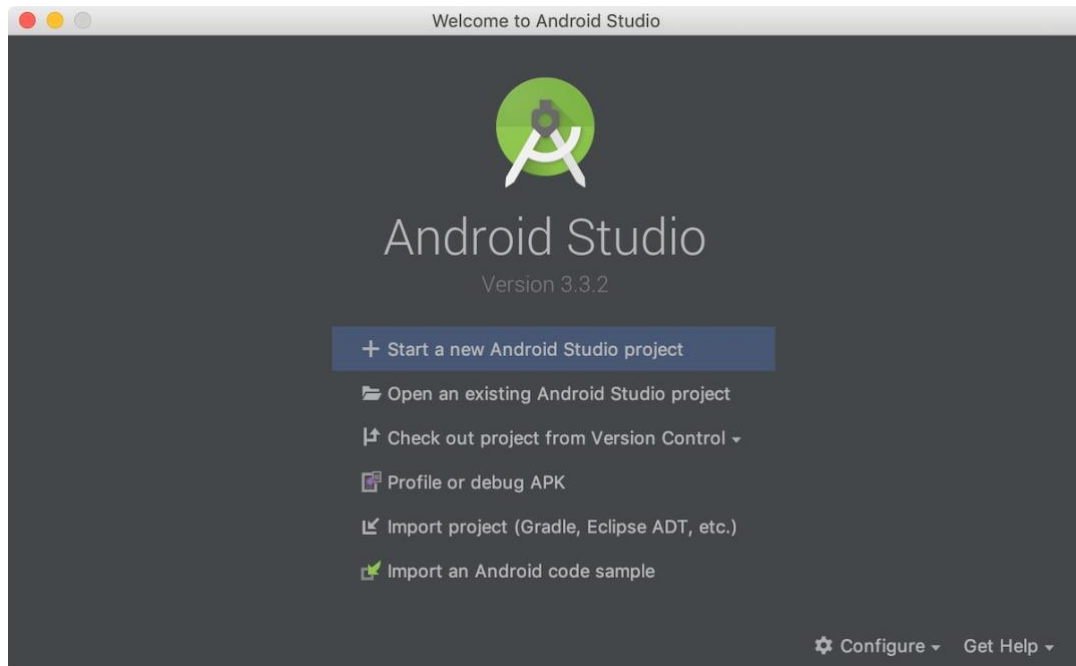
Przykład:

```
var a: Double = 2.15  
val b: Float = 15f  
val c: Long = 20L  
val d: Int = 20  
var e: Short = 15  
var f: Byte = 10  
var g: Char = 'a'  
var h: Boolean = true
```

Więcej o języku Kotlin: <https://kotlinlang.org/>

Tworzymy pierwszą aplikację

- Uruchom aplikację Android Studio I w oknie **Welcome to Android Studio** wybierz **Start a new Android Studio project**.



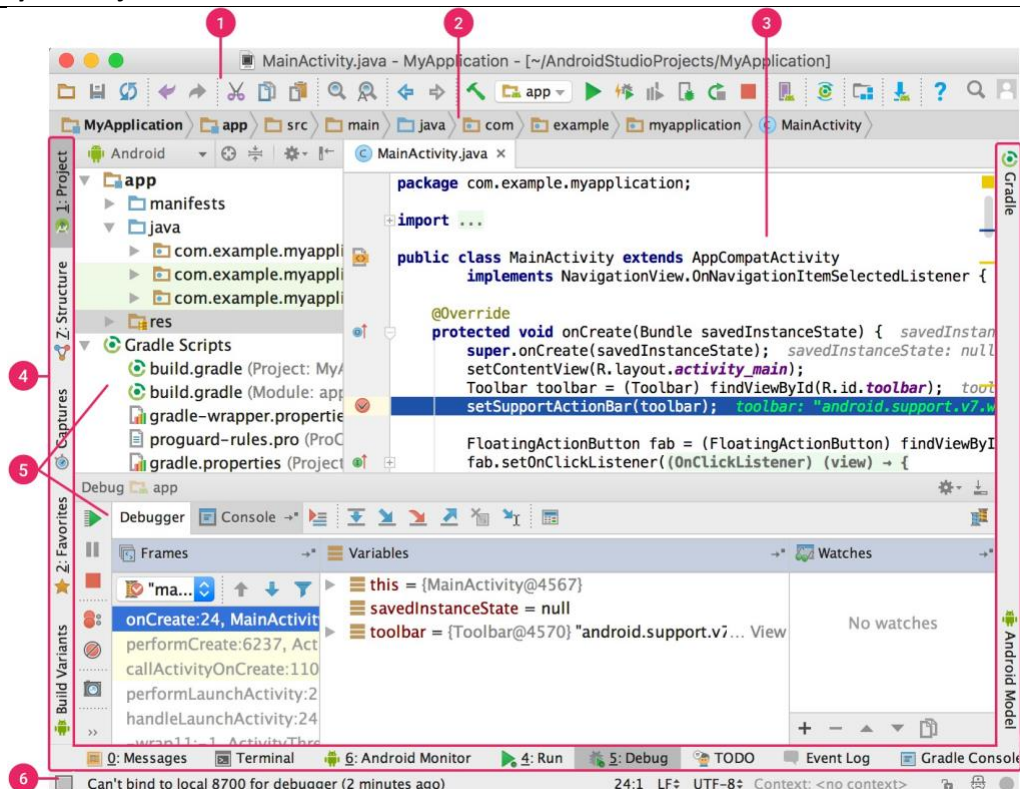
1. W oknie **Choose your project** wybierz **Empty Activity** i kliknij **Next**.
2. W oknie **Configure your project** wpisz następujące wartości
 - **Name:** "My First App"
 - **Package name:** "com.example.myfirstapp"
 - zaznacz **Use AndroidX artifacts**
 - nie zmieniaj lokalizacji domyślnej tworzone aplikacji.
 - Opuść pozostałe opcje.
 - Kliknij **Finish**.

Po wykonaniu tych czynności Android Studio:

- Tworzy folder dla projektu Android Studio o nazwie AndroidStudioProjects.
- Tworzy projekt (może to potrwać kilka chwil).
- Android Studio używa **Gradle** jako swojego systemu budowania aplikacji.

- Otwiera edytor kodu pokazujący twój projekt. Okno Android Studio powinno wyglądać podobnie.
1. **Toolbar** - Pasek narzędzi umożliwia wykonywanie wielu działań, w tym uruchamianie aplikacji i uruchamianie narzędzi z systemem Android.
 2. **Navigation bar** - Pasek nawigacji pomaga poruszać się po projekcie i otwierać pliki do edycji. Zapewnia bardziej zwarty widok widoczny w oknie Projekt.
 3. **Editor window** - Okno edytora służy do tworzenia i modyfikowania kodu. W zależności od bieżącego typu pliku edytor może się zmienić. Na przykład podczas przeglądania pliku układu edytor wyświetla edytor układu.
 4. **Tool window** - Pasek okna narzędzia biegnie wokół zewnętrznej strony okna IDE i zawiera przyciski, które umożliwiają rozwijanie lub zwijanie poszczególnych okien narzędzi.
 5. **Tool windows** - Okno narzędzi zapewnia dostęp do określonych zadań, takich jak zarządzanie projektami, wyszukiwanie, kontrola wersji i inne. Możesz je rozwinąć i zwinąć.
 6. **Status bar** - Pasek stanu wyświetla status projektu i samego IDE, a także wszelkie ostrzeżenia lub komunikaty.

Poznajmy interfejs Android Studio



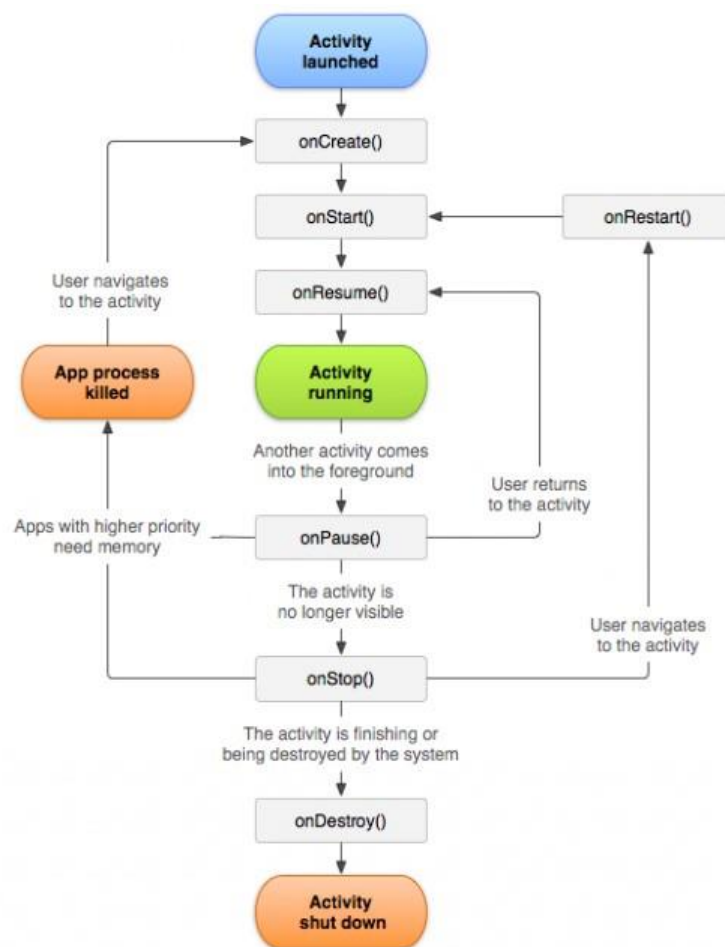
Poznajmy strukturę głównej aktywności

Podstawowym elementem składowym każdej aplikacji mobilnej są Aktywności, które implementują konkretne funkcje programu. Implementacja Aktywności dotyczy zarówno procesów obliczeniowych,

jak i interfejsu komunikacji z użytkownikiem. Każda Aktywność może być utożsamiana z interfejsem użytkownika – obowiązuje zasada takiego projektowania Aktywności, aby jedna Aktywność odpowiadała projektowi jednego „ekranu” interfejsu użytkownika (GUI). Ze względu na podstawową rolę Aktywności wprowadzono różne warianty działania Aktywności – dotyczy to różnych metod wymiany danych oraz różnych sposobów wywoływania Aktywności. Plik

MainActivity.kt odpowiedzialny jest za kod „głównej aktywności”. Użytkownik po zainstalowaniu tego programu na swoim telefonie i uruchomieniu go ujrzy na pierwszym ekranie wszystko to co zostało wygenerowane w metodzie onCreate() klasy MainActivity rozszerzającej klasę Activity.

Dzięki przeładowaniu metody onCreate() system wie jaki layout ma wyświetlić. Aplikacja mobilna może składać się z wielu aktywności (okien). Kluczowe jest pytanie co dzieje się w momencie, w którym przechodzimy z jednej aktywności do drugiej.



Pierwszym etapem cyklu życia aktywności jest jej uruchomienie innymi słowy system Android po uruchomieniu przez użytkownika aplikacji przechodzi do głównej aktywności i ją uruchamia (w ten sposób uruchamiana jest pierwsza – główna aktywność w aplikacji). Następnie czyli już po uruchomieniu aktywności w systemie zachodzą zdarzenia takie jak onCreate(), onStart(), onResume() (dokładnie w takiej kolejności). Dzięki przeładowaniu odpowiednich metod mamy tutaj wpływ na to jak zachowa się nasz program. Na przykład, jak zrobiliśmy to poprzednio możemy, przeładować metodę onCreate() gdzie definiujemy plik xml z layoutem. Kolejnym etapem jest moment, w którym aktywność jest po prostu uruchomiona. Teraz jest czas na obsługę całej interakcji z użytkownikiem, zdarzeń typu onClick itd..

Następnie dochodzimy do najciekawszego fragmentu cyklu, kiedy to mamy trzy możliwości zachowania:

- Inna aktywność innymi działaniami (np. poprzez wciśnięcie przez użytkownika przycisku) wychodzi na pierwszy plan,
- Aktywność (obecna) nie będzie już dłużej widoczna,
- Aktywność (obecna) została zakończona, bądź system ją skasował.

Chodzi teraz o to, że gdy dojdzie do realizacji zachowania na przykład z punktu pierwszego: czyli mamy jakieś tam okno (aktywność) z przyciskiem (wygenerowanym w danym layoucie), który uruchamia inną aktywność po kliknięciu w niego. Kiedy użytkownik klika w ten przycisk (w tym momencie na obecnej aktywności zachodzi zdarzenie `onPause()`) i uruchamia nowego okna. Teraz mamy już zupełnie inną aktywność na pierwszym planie. Gdy w tym momencie klikniemy przycisk „wstecz” z dolnego menu Androida to wrócimy do poprzedniej aktywności ale uwaga (!), nasz kod z przeładowanej metody `onCreate()` nie zostanie wykonany! Jak widać na powyższym obrazku, wracamy w tym wypadku tylko do momentu w którym wykonywana jest metoda `onResume()`.

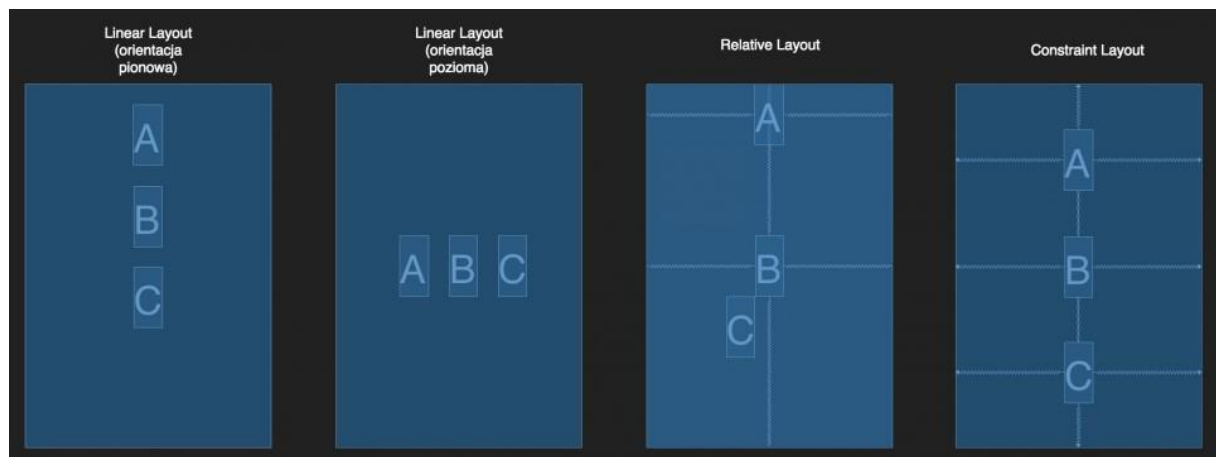
Praca z aktywnościami oraz elementami interfejsu

W tym zadaniu utworzymy układ zawierający pole tekstowe i przycisk. Następnie aplikacja odpowie na przyciśnięcie przycisku, wysyłając zawartość pola tekstowego do innej aktywności.

Layout definiuje wizualną strukturę interfejsu graficznego aplikacji, layouty deklarujemy na dwa sposoby:

- deklaracja w pliku XML
- deklaracja w pliku JAVA

My na razie skupimy się na tworzeniu layoutów za pomocą XML. Wyobraźmy sobie layouty jako kontenery do których wkładamy następne elementy np buttony czy pola tekstowe lub inne kontenery (layouty). Rozróżniamy kilka typów layoutów różnią się one tym jak rozkładane są w nich elementy.



- Linear Layout – najprostszy layout jaki mamy. Linear z angielskiego oznacza „liniowy” – ta nazwa idealnie oddaje jego działanie. Linear Layout po prostu układa elementy jeden za drugim. Wyróżniamy jego dwa warianty – Linear Layout z orientacją pionową, gdzie elementy układane są jeden pod drugim (zaczynając od góry) oraz z orientacją poziomą, gdzie elementy układane są od lewej do prawej.

- Relative Layout – słowo Relative możemy przetłumaczyć jako „względny” lub „zależny” – te dwa słowa również świetnie oddają jego działanie. W tym layoutcie układamy widoki na podstawie położenia innych elementów. Dla przykładu jeśli mamy 3 widoki A, B oraz C to, chcąc ułożyć je jeden pod drugim, musimy powiedzieć systemowi coś w stylu: „Przypnij widok A do góry, widok B do widoku A, a widok C do widoku B”. W ten sposób powstanie nam taka wieża. Relative Layout jest używany w sytuacjach, gdzie chcemy osiągnąć efekt niemożliwy do zrealizowania przy użyciu Linear Layout.
- Constraint Layout – powstał on stosunkowo niedawno – tak naprawdę jest to ulepszony Relative Layout. Każdy widok opisywany jest przez tak zwane constrainty, czyli opis tego w jaki sposób widok ma być przypięty do innego widoku. Na początku nauka Constraint Layouta może wydawać się ciężka, ale Android Studio posiada tryb graficzny, w którym możesz zdefiniować średnio zaawansowany interfejs użytkownika używając jedynie myszki.

W Androidzie praktycznie nie używamy pikseli jako jednostki wielkości. Zamiast tego używamy jednostki „dp” czyli „density independent pixel” co można przetłumaczyć na pixel niezależny od gęstości. Tworząc interfejs deklarujemy na przykład, że nasz przycisk będzie miał 30dp szerokości i Android dba o to, żeby ten przycisk był jednakowej wielkości zarówno na urządzeniu z rozdzielczością HVGA jak i 4K.

Poznajmy strukturę i układ projektu

Spójrz na hierarchię plików dla swojej aplikacji.

1. Kliknij folder **app (1)**, aby rozwinąć hierarchię plików.
2. Kliknij **Project (2)**. Może być konieczne wybranie opcji **View > Tool Buttons** aby zobaczyć tę opcję.
3. Bieżący wybór widoku projektu to Project > Android. Kliknij **Android (3)**, aby przejrzeć inne dostępne widoki projektu.

W widoku Project Android widzisz trzy foldery najwyższego poziomu pod folderem **aplikacji** : **manifest** , **java** i **res** .


1. Rozwiń folder **manifest**. Ten folder zawiera plik AndroidManifest.xml, który opisuje wszystkie składniki aplikacji na Androida i jest odczytywany przez system po uruchomieniu aplikacji.
2. Rozwiń folder **java** . Tutaj organizowane są wszystkie pliki językowe Java. **Java** folder zawiera trzy podfoldery:
 - **com.example.android.myfirstapp** (lub nazwa domeny) : Ten folder zawiera pliki kodu źródłowego Java dla aplikacji.
 - **com.example.android.myfirstapp (androidTest)**: Ten folder jest miejscem, w którym należy umieścić testy instrumentalne.
 - **com.example.android.myfirstapp (test)**: Ten folder jest miejscem, w którym należy umieścić testy jednostkowe.
3. Rozwiń folder **res** . Ten folder zawiera wszystkie zasoby aplikacji, w tym obrazy, pliki układu graficznego, stringi(napisy), ikony i style.
Zawiera następujące podfoldery:

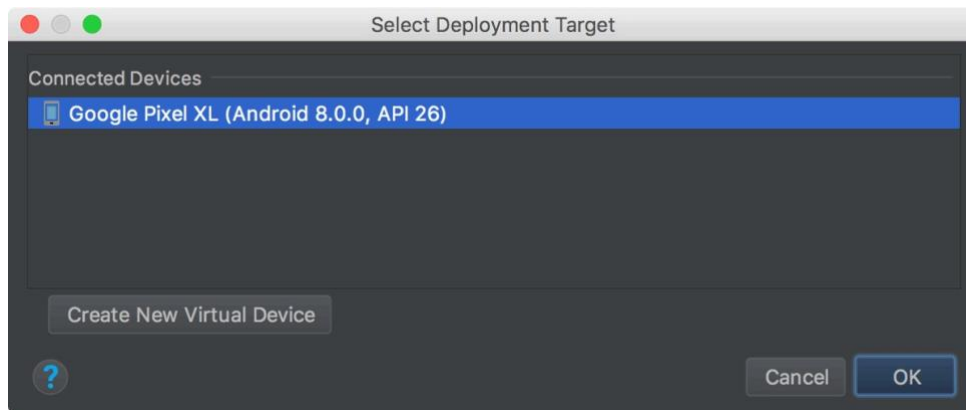
- **drawable** : wszystkie obrazy, pliki graficzne aplikacji będą przechowywane w tym folderze.
 - **layout** : Ten folder zawiera pliki układu graficznego. Obecnie twoja aplikacja ma tylko jeden układ o nazwie activity_main.xml.
 - **mipmap** : ten folder zawiera ikony programu uruchamiającego dla Twojej aplikacji.
 - **values** : zawiera zasoby, takie jak stringi i kolory używane w aplikacji.
4. Rozwiń Gradle Scripts w którym to znajdziesz pliki:
- **build.gradle** - plik zawierający informacje dotyczące kompilacji aplikacji. Można go edytować aby dodać własne moduły, biblioteki czy też zdefiniować miejsce przechowywania kluczy. Jest on integralną częścią projektu.
 - **gradle.properties** - Ustawienia plików „Gradle” (skryptów budujących aplikację).
 - **gradlew** - skrypt startowy „Gradle” dla Unixa.
 - **gradlew.bat** - skrypt startowy „Gradle” dla Windowsa.
 - **settings.gradle** - zawiera informacje o wszystkich pod-projektach jakie muszą zostać skompilowane przy kompilacji aplikacji.

Uruchamiamy aplikację na fizycznym urządzeniu

Android Studio umożliwia uruchamianie aplikacji za pomocą fizycznego urządzenia oraz wirtualnego urządzenia. Na początku sprawdzimy możliwości uruchomienia naszej aplikacji na swoim smartfonie. Skonfiguruj urządzenie w następujący sposób:


1. Podłącz urządzenie do komputera za pomocą kabla USB.
2. Włącz w urządzeniu Android **debugowanie USB** w **opcjach programisty** w następujący sposób.
 1. Otwórz aplikację **Ustawienia**. (Tylko w systemie Android 8.0 lub nowszym) Wybierz **system** .
 2. Przewiń do dołu i wybierz **Informacje o telefonie** .
 3. Przewiń do dołu i dotknij 7 razy **numer kompilacji** .
 4. Wróć do poprzedniego ekranu, aby znaleźć **opcje programisty** w dolnej części.
 5. Otwórz **opcje programisty** , a następnie przewiń w dół, aby znaleźć i włączyć opcję **debugowanie USB** .
3. Uruchom aplikację na swoim urządzeniu w następujący sposób:

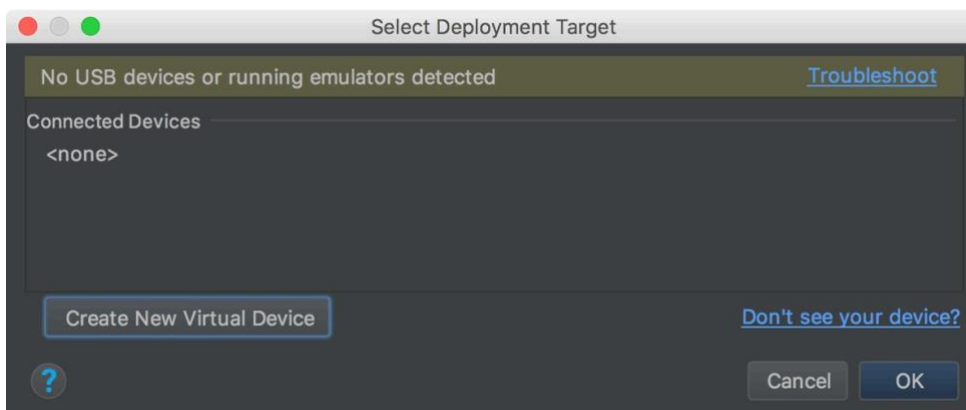
W Android Studio kliknij moduł **app**, w oknie **Project** wybierz **Run > Run** (albo kliknij  na toolbarze).
4. W oknie **Select Deployment Target** wybierz swoje urządzenie i kliknij **OK**. Aplikacja powinna zostać zainstalowana na twoim urządzeniu.



Uruchamiamy aplikację na wirtualnej maszynie:

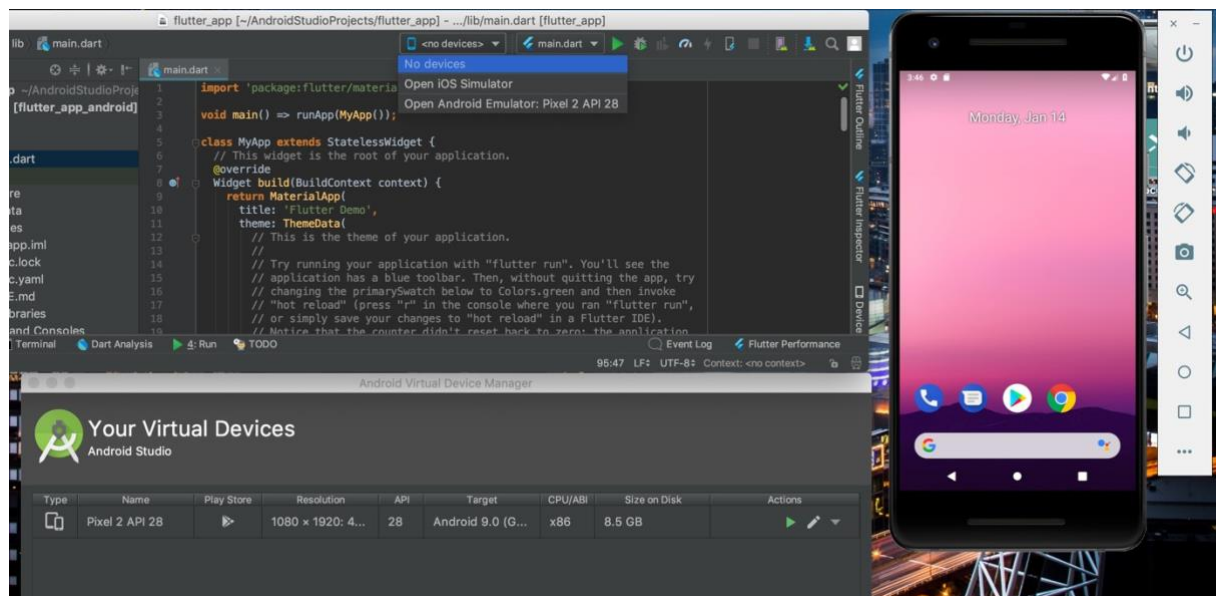
1. Uruchom aplikację w następujący sposób:

W Android Studio kliknij moduł **app**, w oknie **Project** wybierz **Run > Run** (albo kliknij  na toolbarze). W oknie wyboru urządzenia **Select Deployment Target** kliknij **Create New Virtual Device**, jeśli nie masz żadnego zainstalowanego urządzenia.



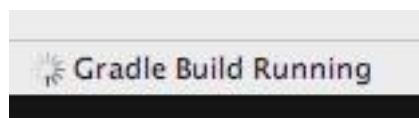
2. W oknie **Select Hardware** wybierz urządzenie takie jak Google Pixel i kliknij **Next**.
3. W oknie **System Image** wybierz wersję z jak największym API Click **Next**. Jeśli nie masz zainstalowanej tej wersji, pojawi się link **Pobierz** , więc kliknij go i zakończ pobieranie. Kliknij **Next** .
4. Na ekranie **Android Virtual Device (AVD)** pozostaw wszystkie ustawienia bez zmian i kliknij **Finish** .

Emulator uruchamia się jak urządzenie fizyczne. W zależności od szybkości komputera może to chwilę potrwać. Możesz sprawdzić też mały poziomy pasek stanu na samym dole Android Studio, aby zobaczyć stan w jakim jest aplikacja.

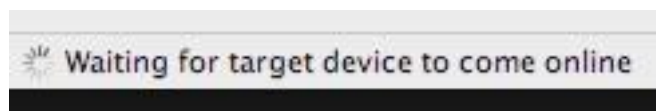


Informacje które mogą pojawić się na krótko na pasku stanu:

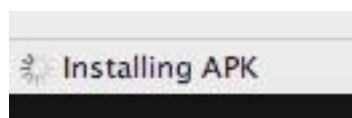
Gradle działa



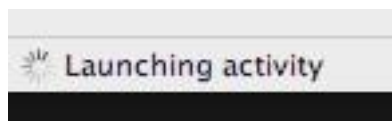
Oczekiwanie na połączenie urządzenia docelowego



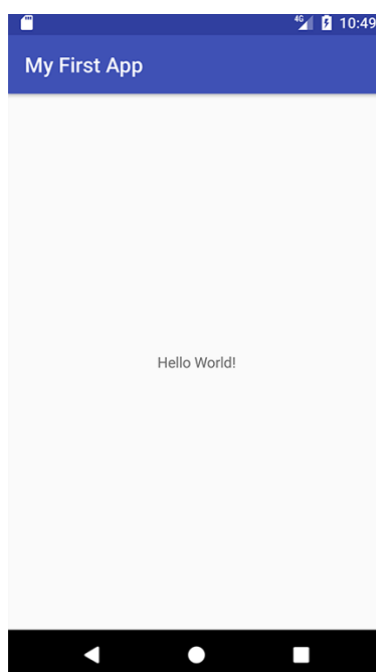
Instalowanie APK



Uruchamianie aktywności




Po utworzeniu aplikacji i przygotowaniu emulatora Android Studio przesyła aplikację do emulatora i uruchamia ją. Dobrą praktyką jest uruchamianie emulatora na początku sesji. Nie zamykaj emulatora, dopóki nie skończysz testować aplikacji, aby nie trzeba było czekać na ponowne uruchomienie emulatora. **Nie należy również uruchamiać więcej niż jednego emulatora jednocześnie, aby zmniejszyć zużycie pamięci.** Po zainstalowaniu aplikacji powinniśmy zobaczyć swoją aplikację, jak pokazano na poniższym zrzucie ekranu.




Pierwsze ćwiczenie:

1. W aplikacji Android Studio w Project window otwórz **app > res > layout > activity_main.xml**.
2. Aby zrobić więcej miejsca dla edytora układu, ukryj okno **projektu**, wybierając **View > Tool**

Windows > Project (albo kliknij **Project**  z lewej strony Android Studio). Jeśli edytor pokazuje kod XML kliknij kartę **Design** na dole ekranu.

3. Kliknij **Select Design Surface**  and select **Blueprint**.

4. Kliknij **Show**  w pasku narzędzi Layout Editor i upewnij się że są zaznaczone **Show Constraints**. Upewnij się, że funkcja Autoconnect jest wyłączona. Podpowiedź na pasku narzędzi

powinno się wyświetlać

Turn On Autoconnect



5. Kliknij

Default

8

Margins

na pasku i ustaw 16.

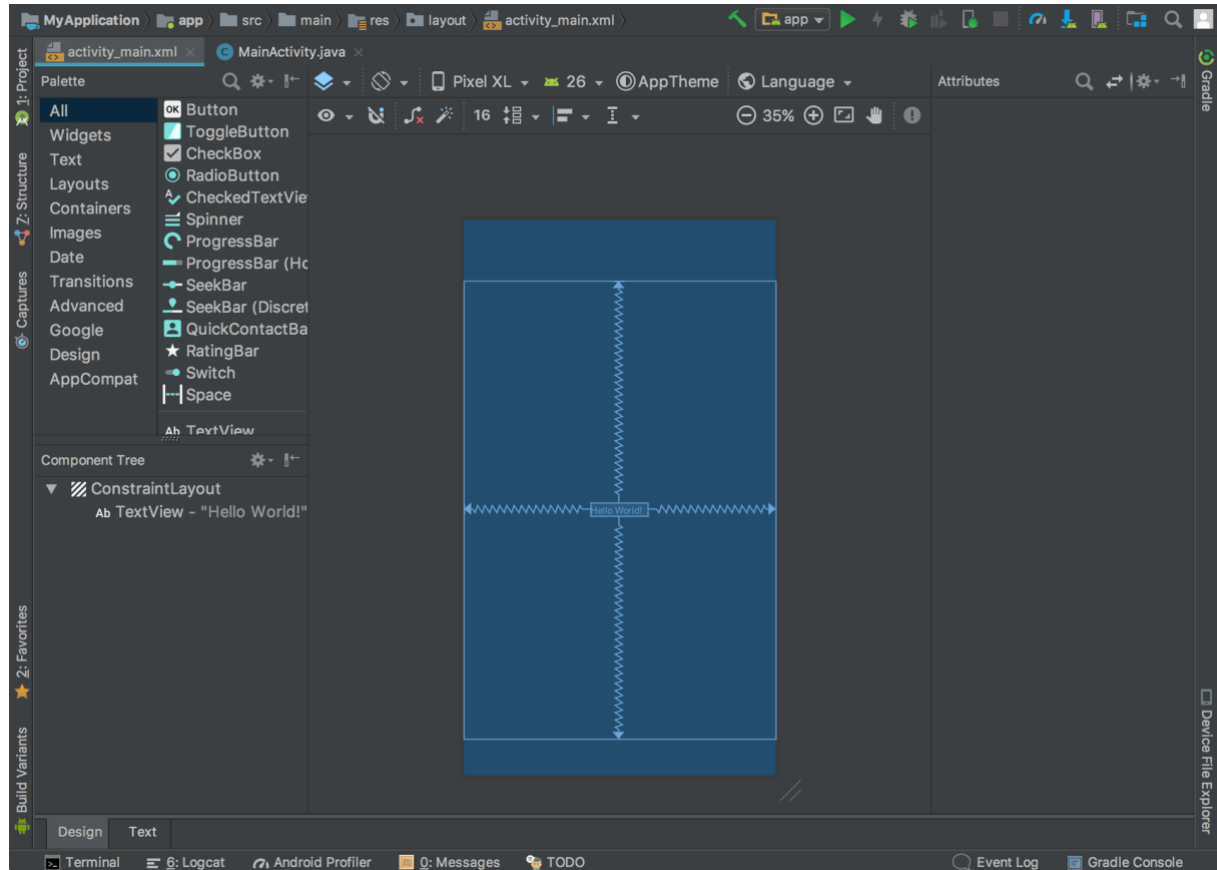
6. Kliknij Device for



Preview

na pasku narzędzi i wybierz 5.5, 1440 × 2560, 560dpi (Pixel XL).

Twój edytor powinien teraz wyglądać tak:



U dołu ekranu mamy dwie zakładki Text i Design:

- Text – to tekstowy tryb tworzenia layoutów za pomocą XML,
- Design – to bardziej przyjazny dla początkujących graficzny interfejs tworzenia layoutów, gdzie po prostu przeciągamy interesujące nas kontrolki i nadajemy im odpowiednie atrybuty.

Umieszczanie komponentów na ekranach aplikacji odbywa się najczęściej poprzez ich przeciągnięcie na ekran (co tak naprawdę generuje kod XML w pliku layoutu) lub ręczne definiowanie ich w pliku layoutu. Poza tym, że figurują jako obiekty XML, nie ma do nich żadnego dowiązania w kodzie aktywności. Jeśli zechcemy w jakiś sposób z tych komponentów korzystać, musimy w klasie aktywności podpiąć do nich referencje.

Okno **Component Tree** w lewym dolnym rogu pokazuje hierarchię widoków układu. W tym przypadku widok główny to ConstraintLayout, zawierający tylko jeden obiekt TextView.

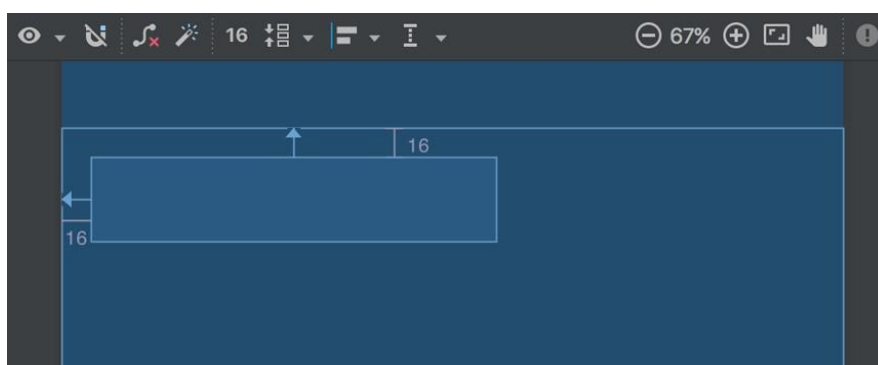
Podstawowe elementy interfejsu z których będziemy korzystać to :

- TextView to pierwszy komponent od góry. Służy do wyświetlania tekstu. Użytkownik aplikacji nie może zmodyfikować jego zawartości. Na palecie komponentów mamy różne jego wariacje

– w tym LargeText, MediumText i SmallText. W rzeczywistości te trzy niby różne elementy to ta sama klasa, ale z różnie ustawionymi parametrami wielkości czcionki. Dostępny jest w palecie w panelu Form Widgets.

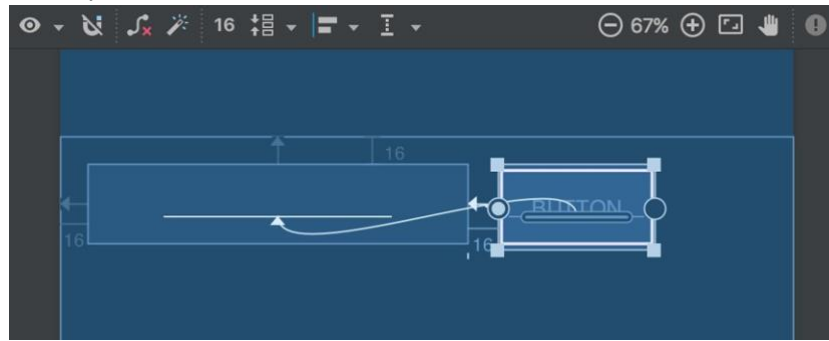
- Button to drugi element od góry. Przycisk. Możemy zaprogramować w Javie sposób jego zachowania po przyciśnięciu. Dostępny jest w palecie w panelu Form Widgets.
- EditText to pole edycyjne w które użytkownik aplikacji może wprowadzać tekst. Dostępny jest w palecie w panelu TextFields. Znajdziemy tam również kilka wariacji tego komponentu, umożliwiających np. wprowadzanie wyłącznie liczb, albo tylko dat. Wszystkie one są obiektami klasy EditText, jednak z różnie ustawionymi parametrami wyświetlania i wprowadzania danych.
- ImageView to element trzeci od dołu. Służy do wyświetlania obrazków. Dostępny jest w palecie w panelu Images & Media.
- ImageButton widoczny jest pod ImageView. Jest czymś pośrodku między Buttonem a ImageView. Możemy na nim wyświetlić obrazek, a jednocześnie oprogramować jego zachowanie po kliknięciu. Dostępny jest w palecie w panelu Images & Media.
- CheckBox umożliwia zaznaczenie lub odznaczenie jakiejś opcji (np. akceptacja regulaminu). Dostępny jest w palecie w panelu Form Widgets.



7. Kliknij **TextView** w oknie **Component Tree** , a następnie naciśnij Delete.
8. W **Palette** kliknij Tekst, aby wyświetlić dostępne kontrolki tekstu.
9. Przeciągnij **Plain Text** do edytora projektu i upuść go w górnej części układu. Jest to widżet EditText, który pozwala na wprowadzanie zwykłego tekstu z klawiatury.
10. Kliknij View w edytorze projektu. Możesz teraz zobaczyć uchwyty zmiany rozmiaru na każdym rogu (kwadraty) i kotwice wiązania po każdej stronie (kółka). Aby uzyskać lepszą kontrolę, możesz powiększyć widok edytora za pomocą przycisków na pasku narzędzi edytora układu.
11. Kliknij i przytrzymaj kotwicę w górnej części, a następnie przeciągnij ją w górę, aż przejdzie na górę układu i zwolnij. To ograniczenie - określa, że widok powinien być 16dp od góry układu (ponieważ marginesy domyślne są ustawione na 16dp). Podobnie, utwórz wiązanie od lewej strony widoku do lewej strony układu.

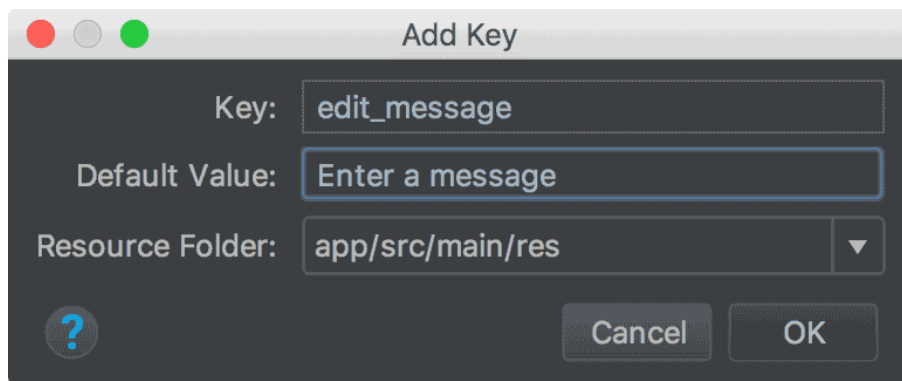



12. W **Palette** kliknij Buttons.
13. Przeciągnij przycisk do edytora projektu i upuść go obok prawej strony.
14. Utwórz wiązanie od lewej strony przycisku po prawej stronie pola tekstowego.
15. Aby ograniczyć widoki w linii poziomej, musisz utworzyć wiązanie między liniami bazowymi tekstu. Tak więc kliknij przycisk, a następnie kliknij **Edit Baseline** , która pojawia się w edytorze projektu bezpośrednio pod wybranym widokiem. Kotwica linii bazowej pojawia się wewnątrz przycisku.


Kliknij i przytrzymaj tę kotwicę, a następnie przeciągnij ją do kotwicy linii bazowej, która pojawia się w polu tekstowym





16. Aby wyświetlić podgląd interfejsu użytkownika, kliknij **Select Design Surface**  na pasku narzędzi i wybierz opcję **Project**. Zauważ, że wprowadzanie tekstu jest wstępnie wypełnione „Name”, a przycisk jest oznaczony „Button”. Teraz zmienimy te napisy.
17. W oknie **Project** wybierz **app > res > values > strings.xml**. Jest to plik zasobów łańcuchowych(stringów), w którym należy zdefiniować wszystkie ciągi interfejsu użytkownika. W ten sposób można zarządzać wszystkimi ciągami interfejsu użytkownika w jednym miejscu, co ułatwia bezpieczeństwem znajdowanie, aktualizowanie i lokalizowanie (w porównaniu z ciągami kodu w układzie lub w kodzie aplikacji).
18. Kliknij **Open editor** w górnej części okna edytora. Spowoduje to otwarcie **Translate Editor**, który zapewnia prosty interfejs do dodawania i edytowania domyślnych ciągów i pomaga w utrzymaniu wszystkich przetłumaczonych łańcuchów.
19. Kliknij **Add Key**  i utwórz nowy string jako "hint text". W oknie **Add Key** wprowadź w Key: "edit_message". Następnie jako domyślną wartość "Enter a message" i kliknij ok. Dodaj kolejny klucz i wprowadź Key: "button_send" z wartością domyślną "Send".



20. Teraz możemy ustawić ciągi dla każdego widoku. Wracamy więc do pliku układu(layout file), klikając plik **activity_main.xml** na pasku kart i dodajemy ciągi w następujący sposób: Kliknij pole tekstowe w układzie, a jeśli okno **Attributes** nie jest już widoczne po prawej stronie, kliknij **Attributes**  na prawym pasku bocznym.

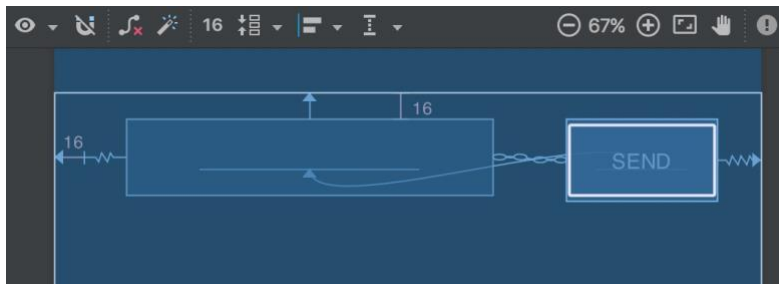
21. Znajdź właściwość **text** obecnie ustawioną na „Name” i usuń wartość. 22. Znajdź właściwość **hint(podpowiedzi)**, a następnie kliknij opcję **Pick a Resource (Wybierz zasób)**  po prawej stronie pola tekstowego. W wyświetlonym oknie dialogowym kliknij dwukrotnie wiadomość **edit_message** z listy. Teraz kliknij przycisk w układzie(layout),

znajdź **text** (obecnie ustawiony na „Button”), kliknij **Pick a Resource (Wybierz zasób)**  , a następnie wybierz **button_send**.

22. Aby utworzyć układ odpowiadający różnym rozmiarom ekranu, teraz rozciągniesz pole tekstowe, aby wypełnić całą pozostałą  przestrzeń poziomą (po uwzględnieniu przycisku i marginesów). Przed kontynuowaniem kliknij **Show** na pasku narzędzi i wybierz **Blueprint**.

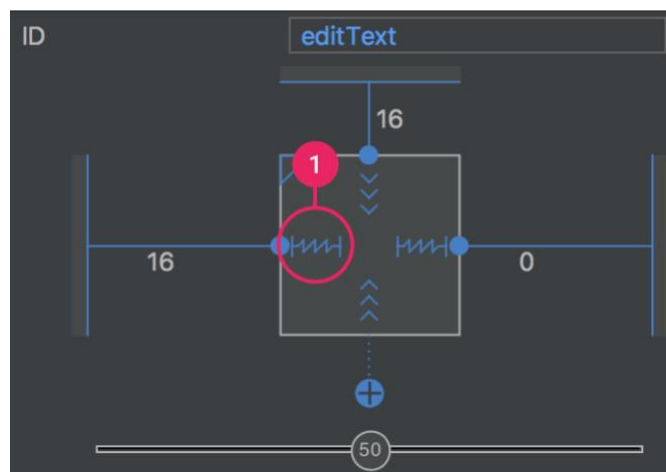
23. Wybierz oba widoki (kliknij jeden, przytrzymaj Shift i kliknij drugi), a następnie kliknij prawym przyciskiem myszy widok i wybierz **Chain > Create Horizontal Chain**

Układ powinien wyglądać tak, jak pokazano na rysunku. Łańcuch jest dwukierunkowym ograniczeniem między dwoma lub więcej widoków

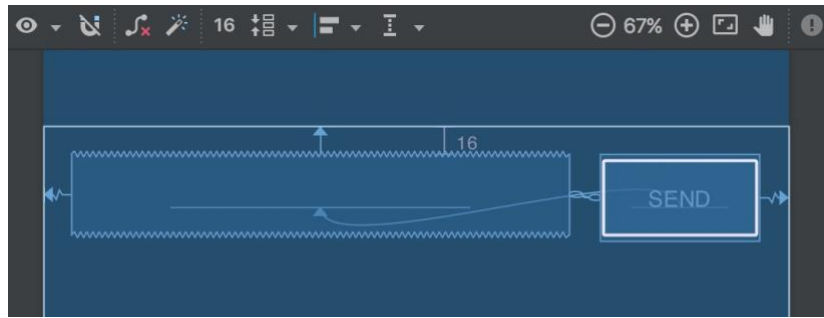


24. Wybierz przycisk i otwórz okno Attributes . Korzystając z inspektora widoku w górnej części okna Attributes , ustaw prawy margines na 16.

25. Kliknij text box, aby wyświetlić jego atrybuty. Kliknij dwukrotnie wskaźnik szerokości, aby ustawić Match Constraints(Dopasowanie ograniczeń), które oznaczają, że szerokość rozszerza się, aby spełnić definicję ograniczeń poziomych i marginesów. Dlatego pole tekstowe rozciąga się, aby wypełnić poziomą przestrzeń (po uwzględnieniu przycisku i wszystkich marginesów).



26. Układ jest gotowy i powinien wyglądać tak jak pokazano na rysunku.



26. Jeśli wygląda na to, że Twój układ nie okazał się zgodny z oczekiwaniami, kliknij poniżej, aby zobaczyć, jak powinien wyglądać Twój XML i porównaj go z tym, co widzisz na karcie **Text**.

Poprawny layout XML:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"    android:layout_height="match_parent"
    tools:context="com.example.myfirstapp.MainActivity">

    <EditText    android:id="@+id/editText"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginTop="16dp"
        android:ems="10"
        android:hint="@string/edit_message"
        android:inputType="textPersonName"
        app:layout_constraintEnd_toStartOf="@+id/button"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="16dp"
        android:layout_marginStart="16dp"
        android:text="@string/button_send"
        app:layout_constraintBaseline_toBaselineOf="@+id/editText"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
```

```

        app:layout_constraintStart_toEndOf="@+id/editText" />
</android.support.constraint.ConstraintLayout>

```

Teraz dodamy kod do MainActivity który uruchomi nowe działanie (nową aktywność), aby wyświetlić wiadomość, gdy użytkownik dotknie przycisk Send .

27. Dodaj metodę do klasy MainActivity wywoływanej przez przycisk w następujący sposób. W **app > java > com.example.myfirstapp > MainActivity**, dodaj metodę **sendMessage()**

```

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

    /** Called when the user taps the Send button */
    fun sendMessage(view: View) {
        // Do something in response to button
    }
}

```

Może pojawić się błąd, ponieważ Android Studio nie może rozpoznać View - klasy użytej jako argument metody. Umieść kursor na View deklaracji metody, a następnie wykonaj szybką poprawkę (Quick Fix), naciskając Alt + Enter (lub Option + Enter na komputerze Mac). (Jeśli pojawi się menu, wybierz select **Import class**)

28. Teraz powracamy do pliku **activity_main.xml** , aby ustawić wywołanie metody z przycisku. Kliknij, aby zaznaczyć przycisk w Layout Editor (edytorze układu).

W oknie **Attributes** znajdź właściwość **onClick** i wybierz **sendMessage [MainActivity]** z listy rozwijanej. Teraz, gdy przycisk zostanie dotknięty, system wywoła metodę sendMessage().

29.Intent to obiekt zapewniający powiązanie środowiska wykonawczego między oddzielnymi komponentami, na przykład dwiema czynnościami. Intent reprezentuje „zamiar coś zrobić.” W naszej aplikacji Intent wykorzystamy do uruchomienia drugiej aktywności. W MainActivity, dodaj stałą EXTRA_MESSAGE i dodatkowy kod do sendMessage() jak pokazano poniżej:

```

const val EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE"

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

    /** Called when the user taps the Send button */
    fun sendMessage(view: View) {
        val editText = findViewById<EditText>(R.id.editText)
        val message = editText.text.toString()
        val intent = Intent(this, DisplayMessageActivity::class.java).apply {

```

```

        putExtra(EXTRA_MESSAGE, message)
    }
    startActivity(intent)
}
}

```

Android Studio ponownie zwróci problem **Cannot resolve symbol** więc wciskamy Alt + Enter (albo Option + Return na Macu). Poprawny import powinien wyglądać tak: `import android.content.Intent;`

```

import androidx.appcompat.app.AppCompatActivity
import android.content.Intent
import android.os.Bundle
import android.view.View
import android.widget.EditText

```

Krótkie wytłumaczenie :

Oto co się dzieje sendMessage():

Intent to konstruktor przyjmujący dwa parametry: Context jako pierwszy parametr (this jest używany, ponieważ Activity jest podklasą Context)

Class to komponent aplikacji, do którego system powinien dostarczyć Intent(w tym przypadku działania, które powinny zostać uruchomiony). putExtra() to metoda , która dodaje z EditText wartość do intencji. Intent


EXTRA_MESSAGE to informacja ponieważ następne działanie używa klucza do pobrania wartości tekstowej. Dobrą praktyką jest definiowanie kluczy dla intencji dodatków przy użyciu nazwy pakietu aplikacji jako prefiksu. Dzięki temu klucze są unikalne, na wypadek, gdyby aplikacja współpracowała z innymi aplikacjami. startActivity() to metoda, która wywołuje DisplayMessageActivity określone przez Intent.

30. Teraz musimy utworzyć tę klasę DisplayMessageActivity. W oknie Project kliknij prawym przyciskiem na folderze app i wybierz New > Activity > Empty Activity. W oknie Configure Activity wpisz "DisplayMessageActivity" do Activity Name i kliknij Finish.

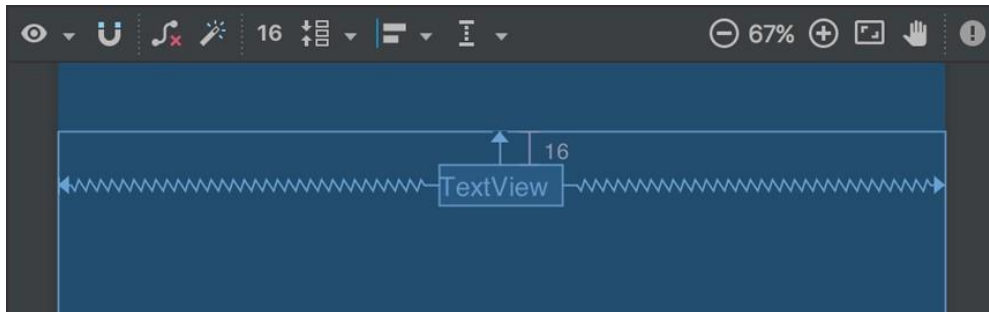
Android Studio automatycznie wykonuje trzy czynności:

- Tworzy DisplayMessageActivityplik.
- Tworzy odpowiedni activity_display_message.xmlplik układu.
- Dodaje wymagany <activity> element w AndroidManifest.xml.

31. Nowa aktywność zawiera pusty plik układu, więc teraz dodamy widok tekstu, w którym pojawi się komunikat. Otwórz **app > res > layout > activity_display_message.xml**. Kliknij **Turn On**

Autoconnect  w pasku narzędzi. W oknie **Palette** wybierz **Text** i przenieś **TextView** na środek w pobliżu górnej środkowej części układu tak, że wskoczy do pionowej linii, która się pojawi.

Automatyczne połączenie dodaje lewe i prawe ograniczenia, aby umieścić widok w centrum poziomym. Utwórz jeszcze jedno ograniczenie od góry widoku tekstu do góry układu, aby wyglądało tak, jak pokazano na rysunku.



Opcjonalnie wprowadź zmiany w stylu tekstu, rozwijając textAppearance w oknie Atrybuty i zmieniając atrybuty, takie jak textSize i textColor .

32. W DisplayMessageActivity dodaj następujący kod do metody onCreate():

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

    super.onCreate(savedInstanceState)

    setContentView(R.layout.activity_display_message)


    // Get the Intent that started this activity and extract the string
    val message = intent.getStringExtra(EXTRA_MESSAGE)


    // Capture the layout's TextView and set the string as its text
    val textView = findViewById<TextView>(R.id.textView).apply {

        text = message

    }
}
```

Naciśnij Alt + Enter (lub Option + Return na Macu), aby zaimportować brakujące klasy. Import powinien wyglądać następująco:

```
import androidx.appcompat.app.AppCompatActivity
import android.content.Intent
import android.os.Bundle
import android.widget.TextView
```

33. Dodajmy pasek nawigacji do naszej aplikacji. Każdy ekran w aplikacji, który nie jest głównym punktem wejścia (wszystkie ekrany, które nie są ekranem „domowym”) powinien zapewniać nawigację, aby użytkownik mógł powrócić do ekranu nadrzędnego w hierarchii aplikacji(np. poprzedniego), naciskając przycisk na pasku aplikacji.

Otwórz plik `app> manifests> AndroidManifest.xml` , znajdź `<activity>` w `DisplayMessageActivity` i zastąp go następującym:

```
<activity android:name=".DisplayMessageActivity"
android:parentActivityName=".MainActivity">
    <!-- The meta-data tag is required if you support API level 15 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

System Android teraz automatycznie doda przycisk **W górę** na pasku aplikacji.

34. Teraz uruchom aplikację ponownie, klikając przycisk **Apply Changes** ⚡ na pasku narzędzi. Lub kliknij **Uruchom**, ▶ aby zainstalować i uruchomić aplikację. Po otwarciu wpisz wiadomość w polu tekstowym i dotknij przycisk **Send** , aby zobaczyć wiadomość w drugim działaniu.



Ikony w Android Studio:

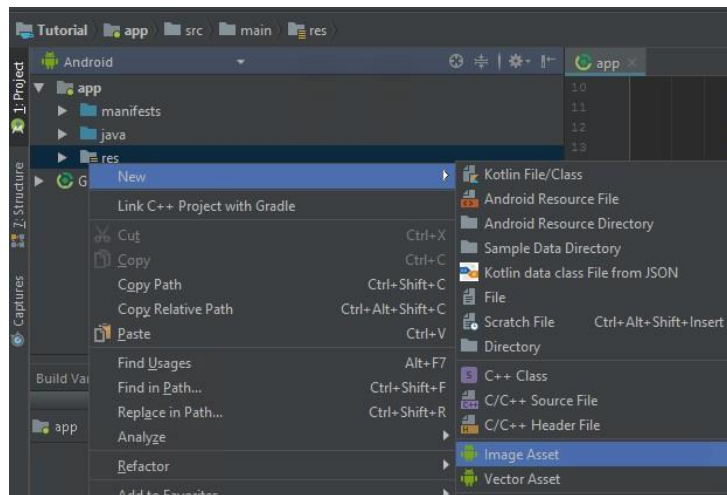
Ikony w Androidzie podzielone są na kilka kategorii. W tym temacie zajmiemy się ustawianiem "głównej" ikony aplikacji - **Launcher Icon**.

Posiada ona dwa podtypy:

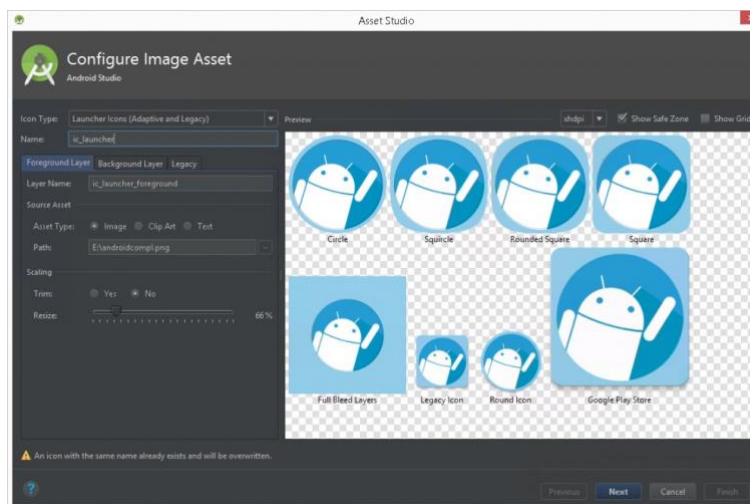
- Legacy Launcher Icon - Android 7.1 (API 25) i niższe
- Adaptive Launcher Icon - dla Androida 8.0 (API 26) i wyższe.

Aby stworzyć Launcher Icon najłatwiej użyć narzędzia wbudowanego w Android Studio jakim jest **Image Asset Studio**.

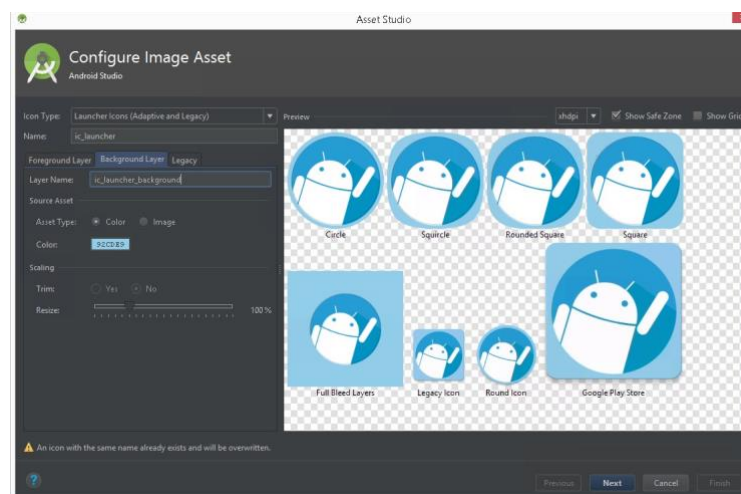
1. W folderze `res` w naszym projekcie klikamy prawym przyciskiem myszy. Wybieramy **New** -> **Image Asset**:



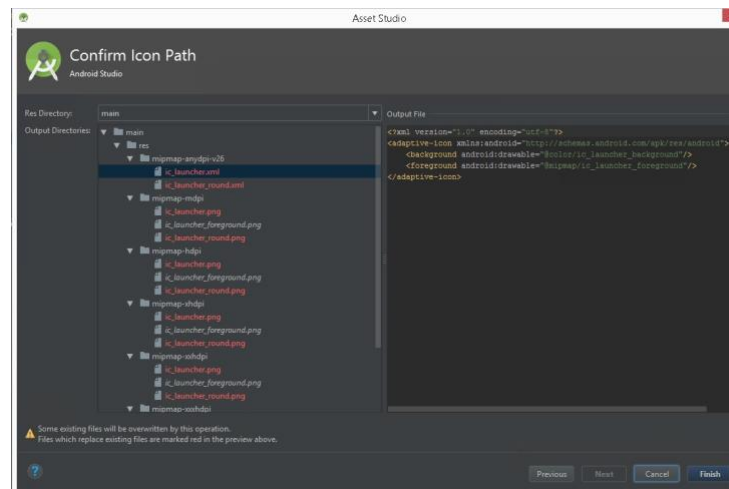
2. Domyślnie otwiera nam się kreator do utworzenia obu typów ikon. Jeżeli posiadamy własny obrazek wybieramy Asset Type: Image i podajemy ścieżkę do pliku. Zmniejszyć można ikonę za pomocą suwaka przy Resize, by mieściła się w Safe Zone.



3. Przejdź do zakładki **Background Layer** i ustaw kolor, tak aby pasował do ikony:



- Następnie klikamy **Next** i na następnym ekranie widzimy jakie pliki i w jakiej lokalizacji zostaną utworzone



- Na końcu klikamy **Finish**. Po przebudowaniu i ponownej instalacji na urządzeniu/emulatorze, nasza aplikacja będzie posiadać nową ikonę.
- W tym przykładzie dodaj własną ikonę, którą stworzysz w dowolnym programie graficznym i zaimportujesz dla poprzedniej aplikacji.

Toast – pierwszy sposób na powiadomienie

Toast w Androidzie służy do wyświetlania fragmentu tekstu przez krótki okres czasu. Po utworzeniu Toastu tekst pojawia się na ekranie, pozostaje na nim przez około 2–3 do 5 sekund i znika.

Toast występuje w dwóch wersjach z opcją LENGTH_SHORT (dla krótkiego tekstu) oraz LENGTH_LONG (dla dłuższego tekstu).

```
Toast.makeText(this, "Hello World! To jest Toast.", Toast.LENGTH_SHORT).show()
```

```
Toast.makeText(this, " Hello World! To jest Toast. Wnieśmy Toast za nasze działanie. Za nasze programowanie! ", Toast.LENGTH_LONG).show()
```

W tym zadaniu wyjaśnię również jak działa przycisk oraz OnClickListener dla przycisku. Przycisk(buton) to element interfejsu użytkownika, który zwykle służy do odbierania działań użytkownika jako danych wejściowych.

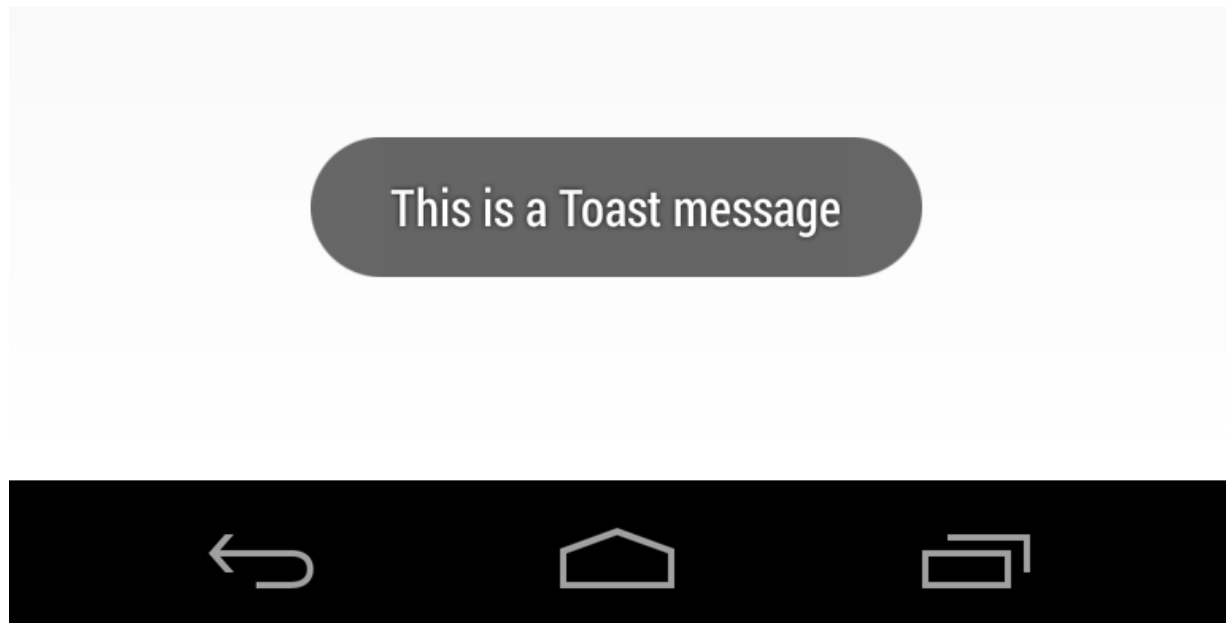
Przykładowy zapis

```
// get reference to button
val btn_click_me = findViewById(R.id.btn_click_me) as Button
// set on-click listener
btn_click_me.setOnClickListener {
    Toast.makeText(this@MainActivity, "This is a Toast messeage.", Toast.LENGTH_SHORT).show()
}
```

To, co zrobiliśmy tutaj, to odwołanie do przycisku : **val btn_click_me = findViewById(R.id.btn_click_me) as Button**. Utworzyliśmy zmienną o nazwie val btn_click_me do

której powiązany został przycisk o nazwie `btn_click_me`. Następnie zastosowaliśmy metodę `setOnClickListener` do uruchomienia akcji po kliknięciu przycisku.

Poniżej możemy zobaczyć jak wygląda Toast w praktyce. Czy znasz go z jakiś aplikacji ?



1. Utwórz nowy project z klasą `MainActivity.kt` w której umieścisz na środku ekranu/aktywności przycisk(buton). Po naciśnieniu w niego aplikacja powinna zwrócić powiadomienie Toast.
2. Przycisk powinien nazywać się „**btn_click_me**”
3. Przetestuj działanie Toast’a dla krótkiego i dłuższego tekstu.
4. Spróbuj dodać kilka innych przycisków oraz powiadomień Toast.
5. W wersji trudniejszej spróbuj utworzyć `SecondActivity` oraz powiąż ją z przyciskiem za pomocą Intencji.
6. Przykładowo plik `MainActivity.kt` powinien wyglądać tak :

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_toast)  
  
        var btn_click_me = findViewById(R.id.btn_click_me) as Button  
  
        btn_click_me.setOnClickListener {  
            // make a toast on button click event  
            Toast.makeText(this, "Hi there! This is a Toast.", Toast.LENGTH_LONG).show()  
        }  
    }  
}
```

7. Oczywiście należałoby wykonać również import zależności :

```
import android.support.v7.app.AppCompatActivity  
import android.os.Bundle  
import android.widget.Button
```

```
import android.widget.Toast
```

8. Przykładowy plik xml powinien wyglądać tak:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.devillecloud.myapplication.MainActivity">
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:padding="25dp"
    android:orientation="horizontal">
    <Button
        android:id="@+id/btn_click_me"
        android:text="Wciśnij mnie"
        android:textAllCaps="false"
        android:padding="10dp"
        android:textSize="25dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    </LinearLayout>
</android.support.constraint.ConstraintLayout>
```

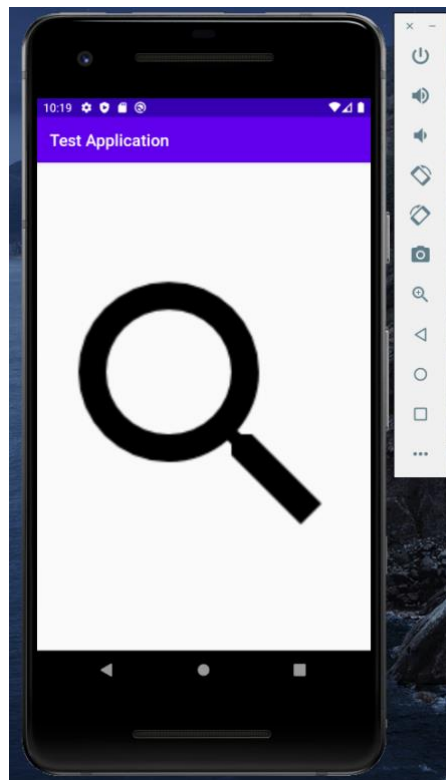
ImageView – obrazki w Aktywnościach :

W Androidzie ImageView jest klasą potomną View, a zatem metoda `setOnClickListener ()` może być użyta na obiekcie typu `ImageView`. Gdy `OnClickListener` jest ustawiony na `ImageView`, kliknięcie na `ImageView` wyzwala akcję i kod w metodzie `setOnClickListener` jest wykonywany.

Przykład:

```
// get reference to ImageView
val image_view = findViewById(R.id.image_view) as ImageView
// set on-click listener for ImageView
image_view.setOnClickListener {
    // your code here
    Toast.makeText(this@MainActivity, "You clicked on ImageView.", Toast.LENGTH_SHORT).show()
}
```

Przykładowy wygląd aplikacji można zobaczyć na poniższym rysunku:



1. Utwórz nowy projekt w którym umieścisz zdjęcie/obrazek. W tym programie po kliknięciu w obrazek powinno pokazać się powiadomienie typu Toast.
2. Zastosuj w programie powyższy kod. Obrazek powinien nazywać się image_view.
3. Przykładowy zapis programu powinien wyglądać tak:

```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.widget.ImageView
import android.widget.Toast

/**
 * An example to set OnClickListener for ImageView in Kotlin Android
 */
class MainActivity : AppCompatActivity() {

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

        // get reference to ImageView
        val iv_click_me = findViewById(R.id.iv_click_me) as ImageView
        // set on-click listener
        iv_click_me.setOnClickListener {
            // your code to perform when the user clicks on the ImageView
        }
    }
}
```

```

        Toast.makeText(this@MainActivity, "Hello World.", Toast.LENGTH_SHORT).show()
    }
}
}

```

4. Przykład kodu XML.

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.devillecloud.myapplication.MainActivity">
    <LinearLayout
        android:id="@+id/ll_main_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:orientation="vertical">
        <ImageView
            android:id="@+id/iv_click_me"
            android:src="@drawable/image1"
            android:layout_gravity="center"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
        </LinearLayout>

    </android.support.constraint.ConstraintLayout>

```

5. Eksperymentuj z tą aplikacją. Zaproponuj umiejscowienie TextView oraz Button w swojej aplikacji.