

Kotlin & Android w Android Studio Giraffe (2022.3.1)

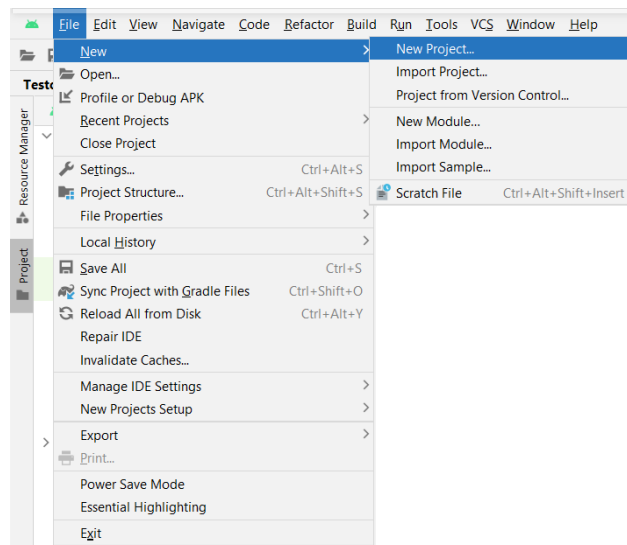
Wprowadzenie

Kotlin – statycznie typowany język programowania działający na maszynie wirtualnej Javy, 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.

Tworzenie nowego projektu z wykorzystaniem szablonu

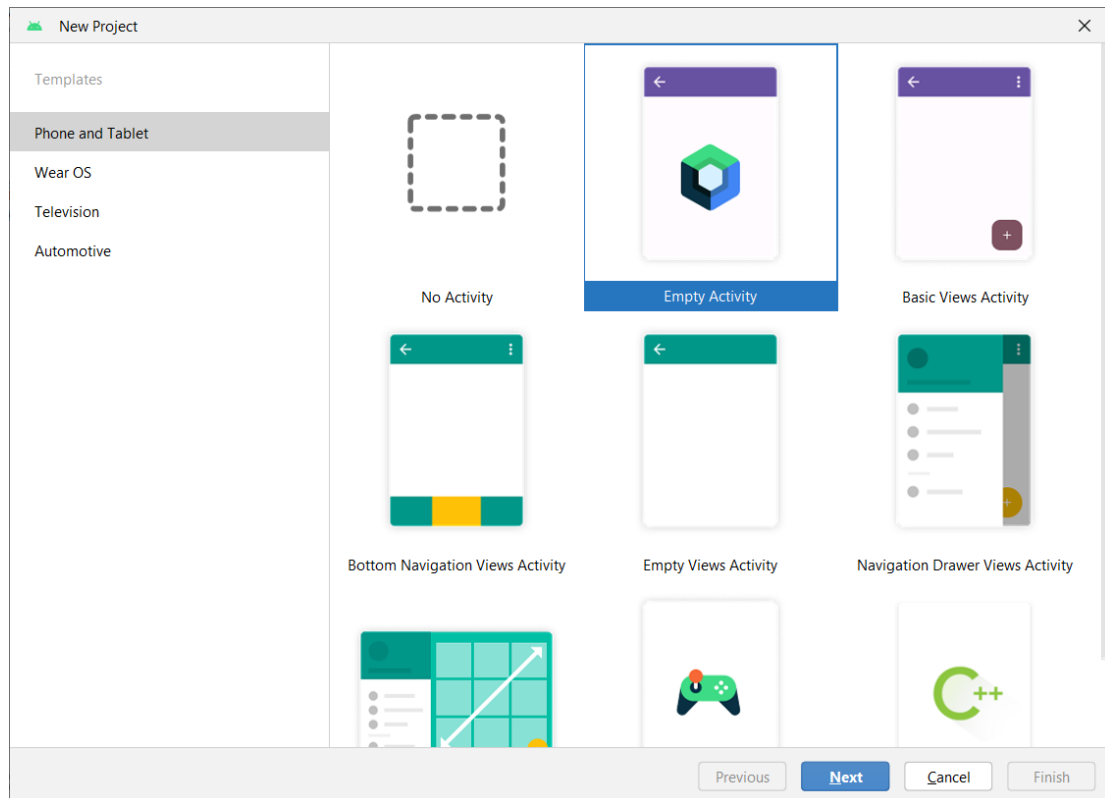
Nowy projekt

W celu utworzenia nowego projektu AndroidStudio wybieramy kolejno **File/New/New Project**:



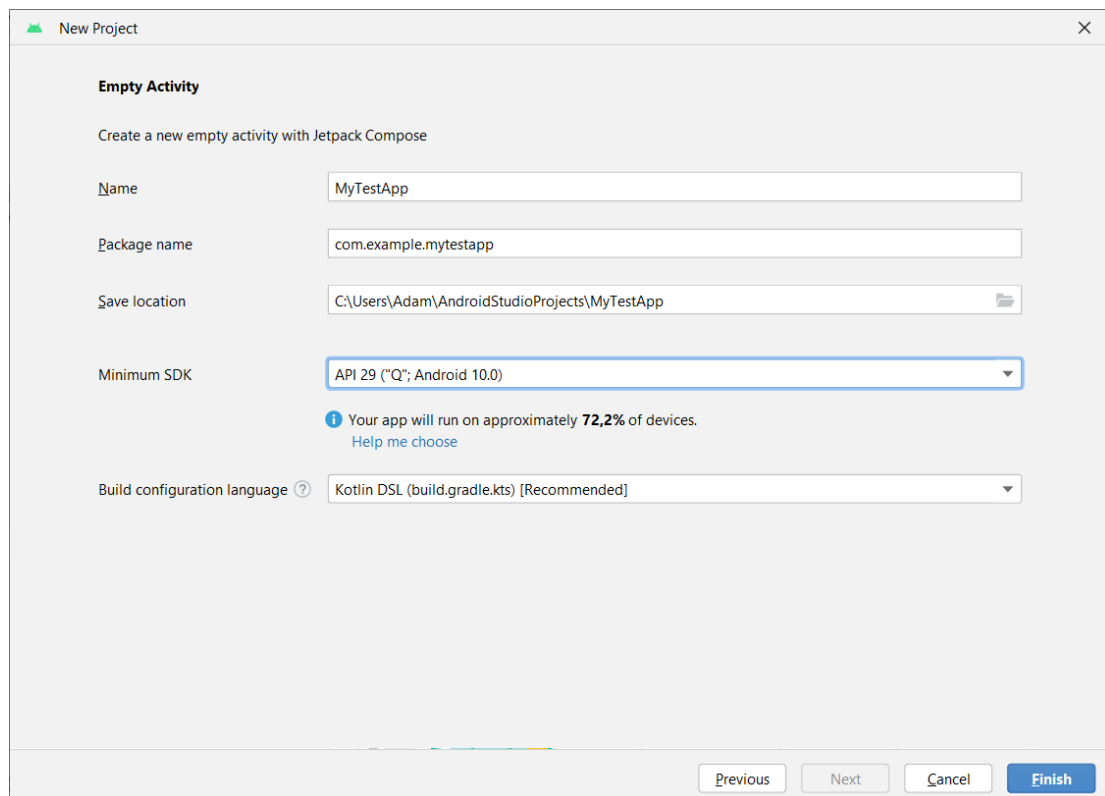
Otworzy się okno *Nowy projekt* z listą szablonów dostarczonych przez Android Studio. Szablony tworzą strukturę projektu i pliki potrzebne Android Studio do zbudowania projektu. Wybrany szablon zawiera kod startowy, dzięki któremu możemy szybciej działać.

Dla celów tego laboratorium wybieramy **Empty Activity** w sekcji **Phone and Tablet**. Szablon *Empty Activity* umożliwia utworzenie prostego projektu, pozwalającego na zbudowanie aplikacji posiadającej jeden ekran i wyświetlającej tekst „Hello Android!”.



Klikamy **Next** i pojawi się okno, gdzie konfigurujemy nasz projekt. Wprowadzamy nazwę aplikacji, nazwę pakietu (pozostawiamy bez zmian), wybieramy lokalizację, minimalną wersję SDK dla naszej aplikacji (API 29 Android 10.0) oraz język programowania (zaznaczamy Kotlin). Pole *Minimum SDK* wskazuje minimalną wersję Androida, na której może działać nasza aplikacja.

Następnie klikamy **Finish**.



W tym momencie następuje tworzenie projektu, jego struktury i potrzebnych plików. Proces ten może chwilę potrwać.

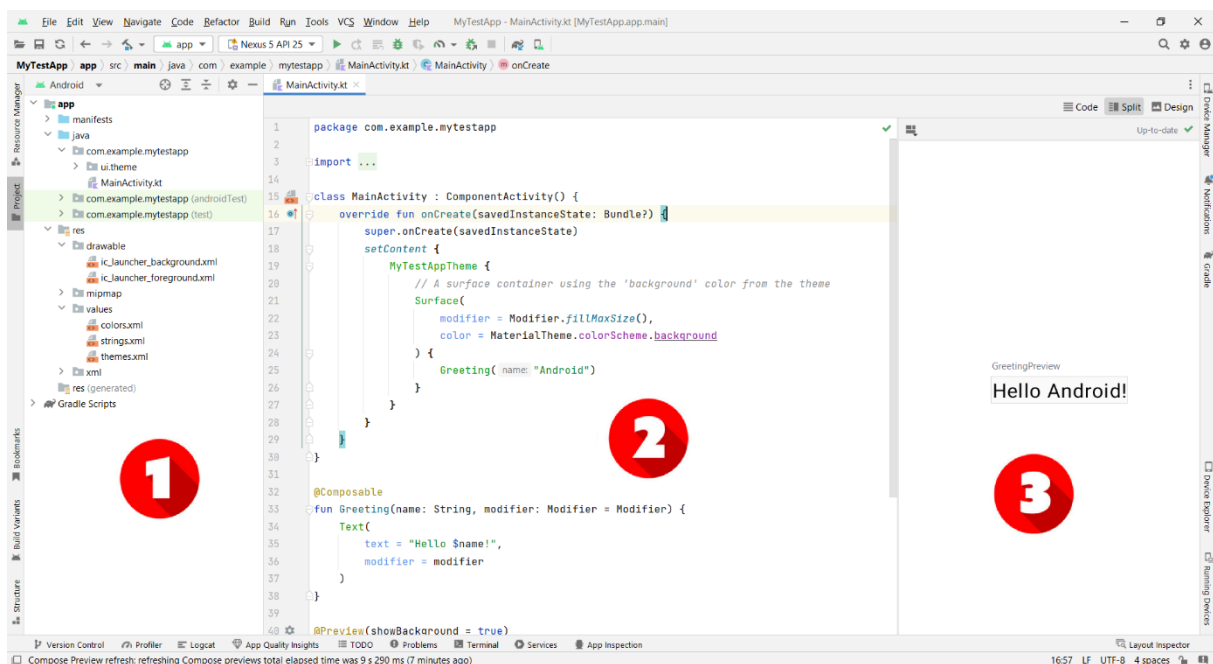
Okno projektu

W prawej górnej części okna Android Studio mamy dostępne 3 kafelki:

Code Split Design

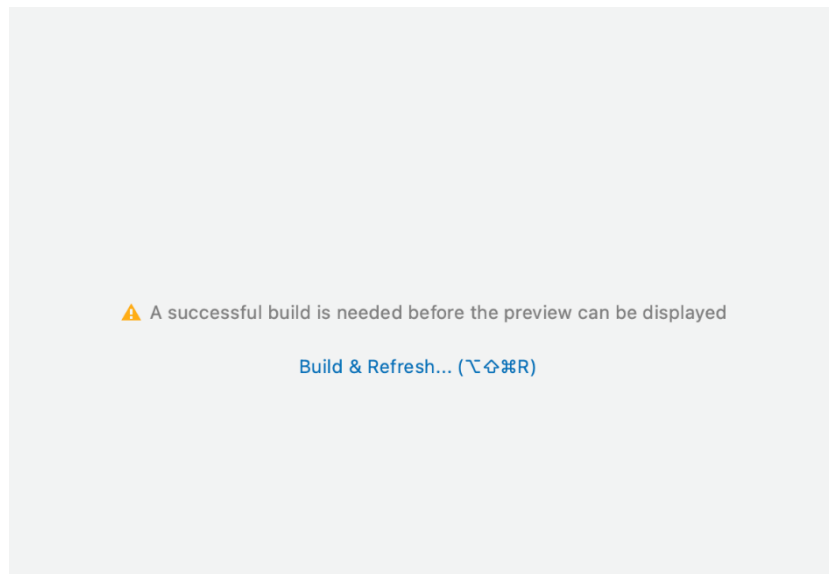
Klikając opcję *Split* możemy wyświetlić zarówno kod, jak i wygląd projektu. Możemy także kliknąć *Code*, aby wyświetlić tylko kod, lub kliknąć *Design*, aby wyświetlić tylko wygląd.

Wybierając *Split* okno Android Studio podzielone zostanie na 3 obszary:

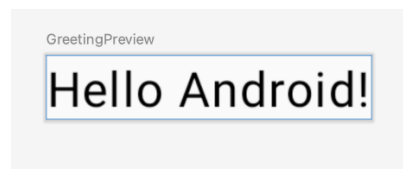


- Widok *Project* (1) pokazuje pliki i foldery projektu
- Widok *Code* (2) to miejsce, w którym edytujemy kod
- Widok *Design* (3) umożliwia podgląd wyglądu aplikacji

W części *Design* widzimy puste okienko z następującym tekstem:

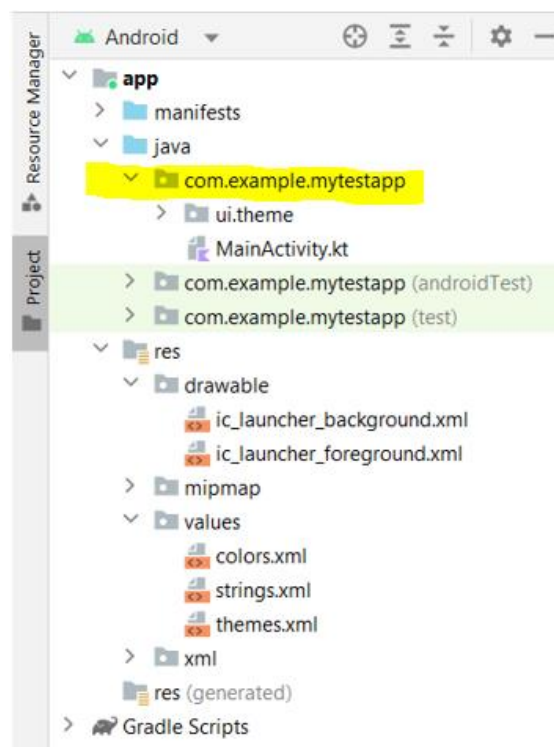


Klikamy opcję *Build & Refresh*. Kompilacja może zająć trochę czasu, ale po jej zakończeniu w podglądzie zostanie wyświetlone pole tekstowe z napisem „Hello, Android!”.



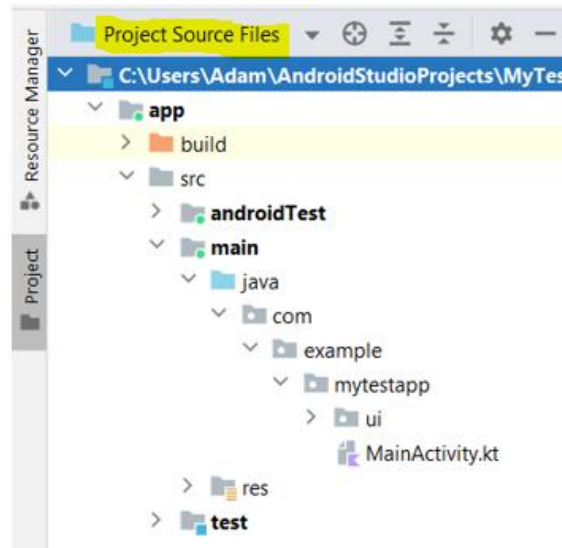
Struktura projektu

W Android Studio w zakładce *Project* wyświetlane są pliki i foldery naszego projektu. Kiedy konfigurowaliśmy projekt, nazwa pakietu brzmiała *com.example.mytestapp*. Pakiet ten możemy zobaczyć w zakładce *Project*.

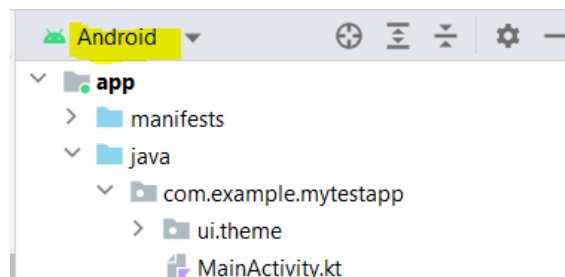


Jest to standardowy widok i organizacja plików, przydatna podczas pisania kodu dla projektu. Łatwo można uzyskać dostęp do plików, nad którymi będziemy pracować w aplikacji. Jeśli jednak spojrzymy na pliki w przeglądarce plików (np. Eksplorator Windows), to ich hierarchia jest zorganizowana zupełnie inaczej.

W Android Studio możemy w łatwy sposób uzyskać taki widok projektu wybierając *Project Source Files* z menu rozwijanego zakładki *Project*.

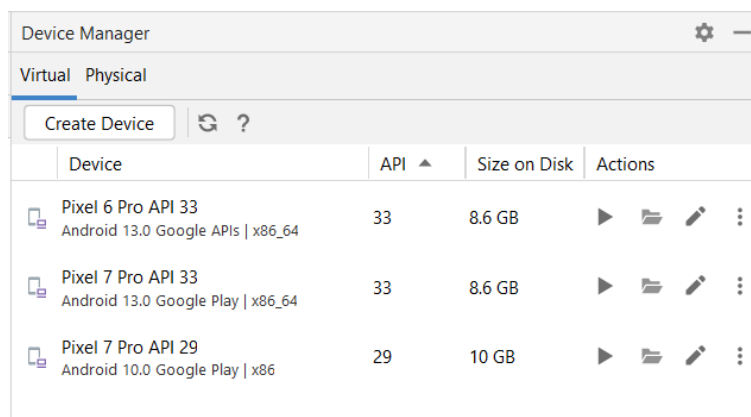


Aby wrócić do poprzedniego widoku wybieramy ponownie opcję *Android* z menu rozwijanego zakładki *Project*.

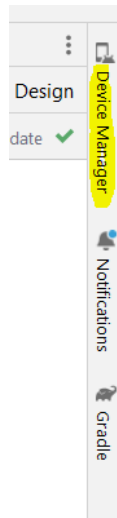


Uruchomienie aplikacji

Aby uruchomić aplikację skorzystamy z emulatora. W tym celu wybieramy z menu Android Studio kolejno: **Run/Select Device...**, a następnie **Device Manager**. Wywołany zostanie manager urządzeń.



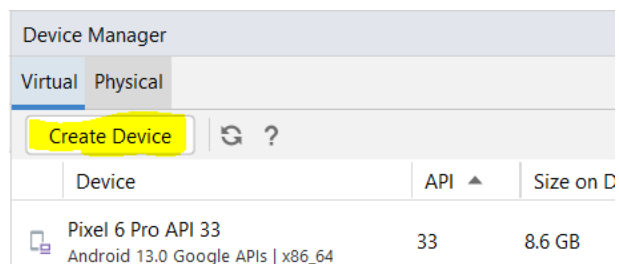
Manager urządzeń może zostać wywołany również poprzez kliknięcie na *Device Manager* w prawej części okna Android Studio.



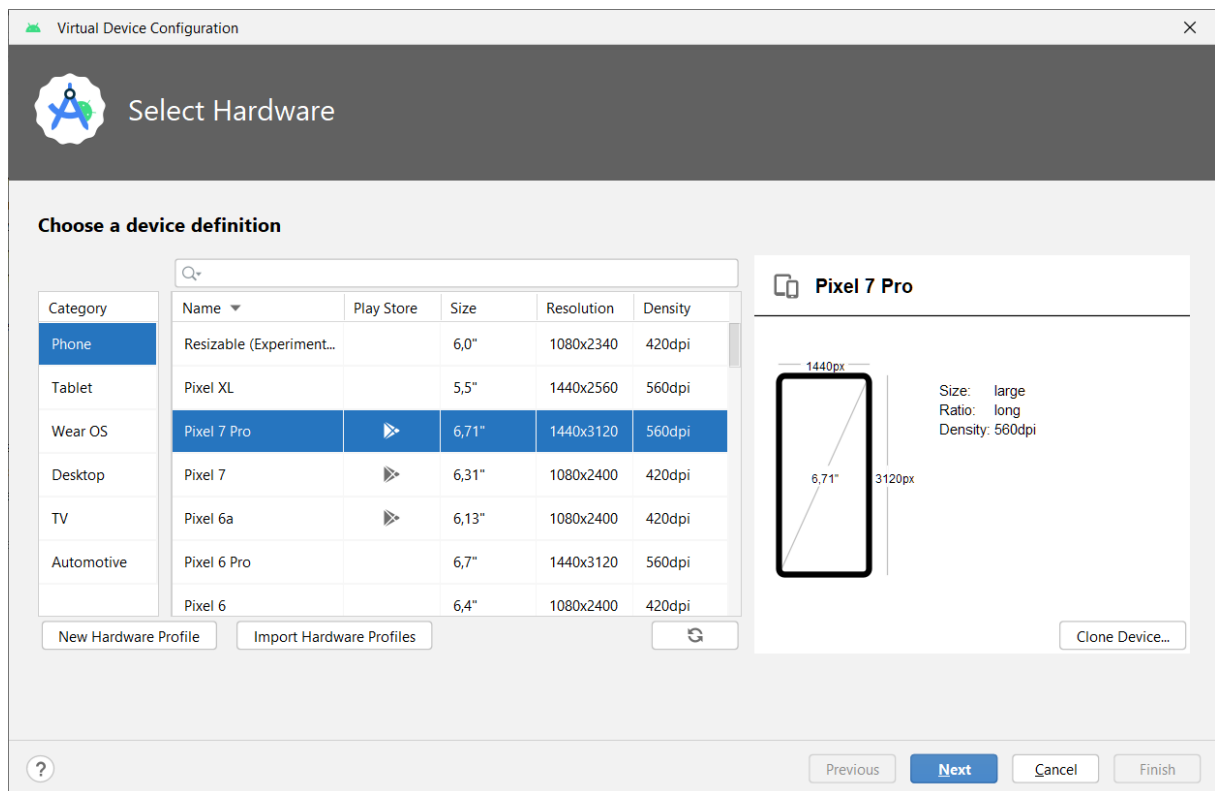
Manager zawiera 2 zakładki: *Virtual* i *Physical*, odpowiednio dla urządzeń wirtualnych (emulatorów) i urządzeń fizycznych (smartfonów, tabletów itp.).

Uruchamianie aplikacji na urządzeniu wirtualnym (emulatorze)

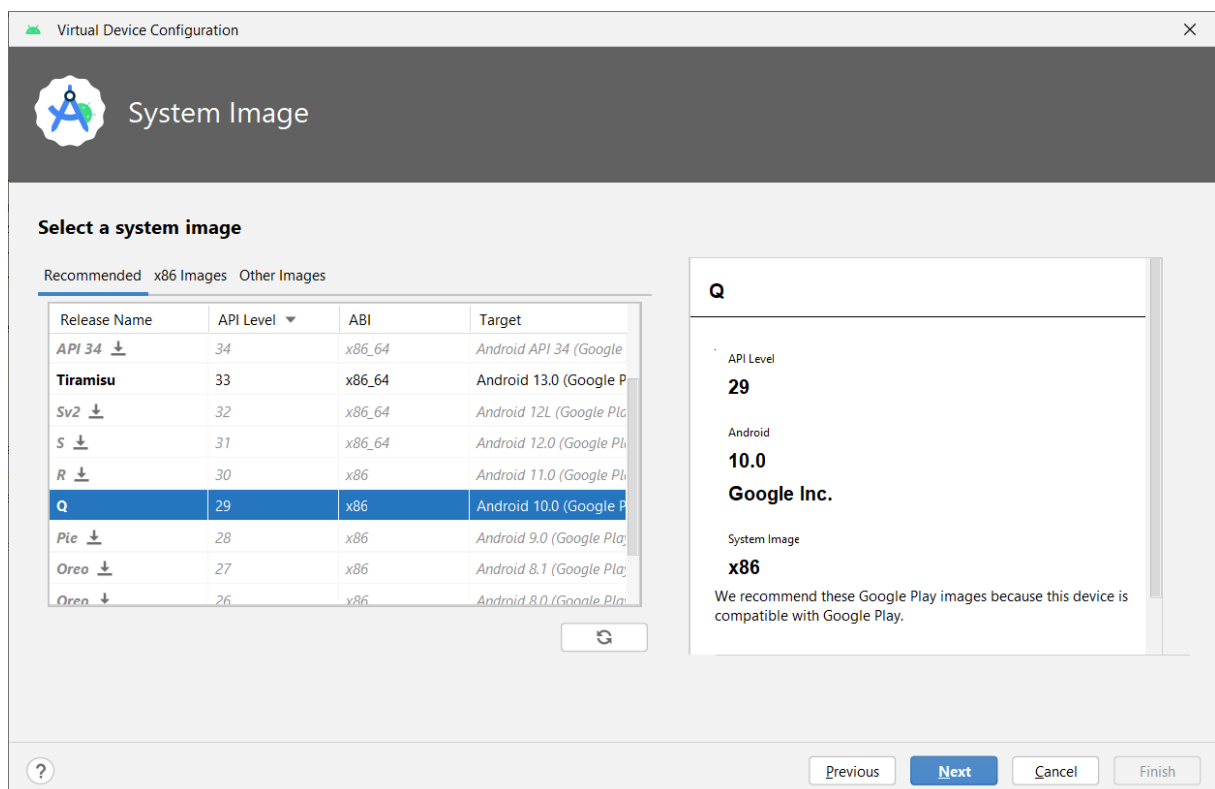
W zakładce *Virtual* mamy listę zainstalowanych emulatorów oraz możemy dodać kolejne klikając na *Create Device*.



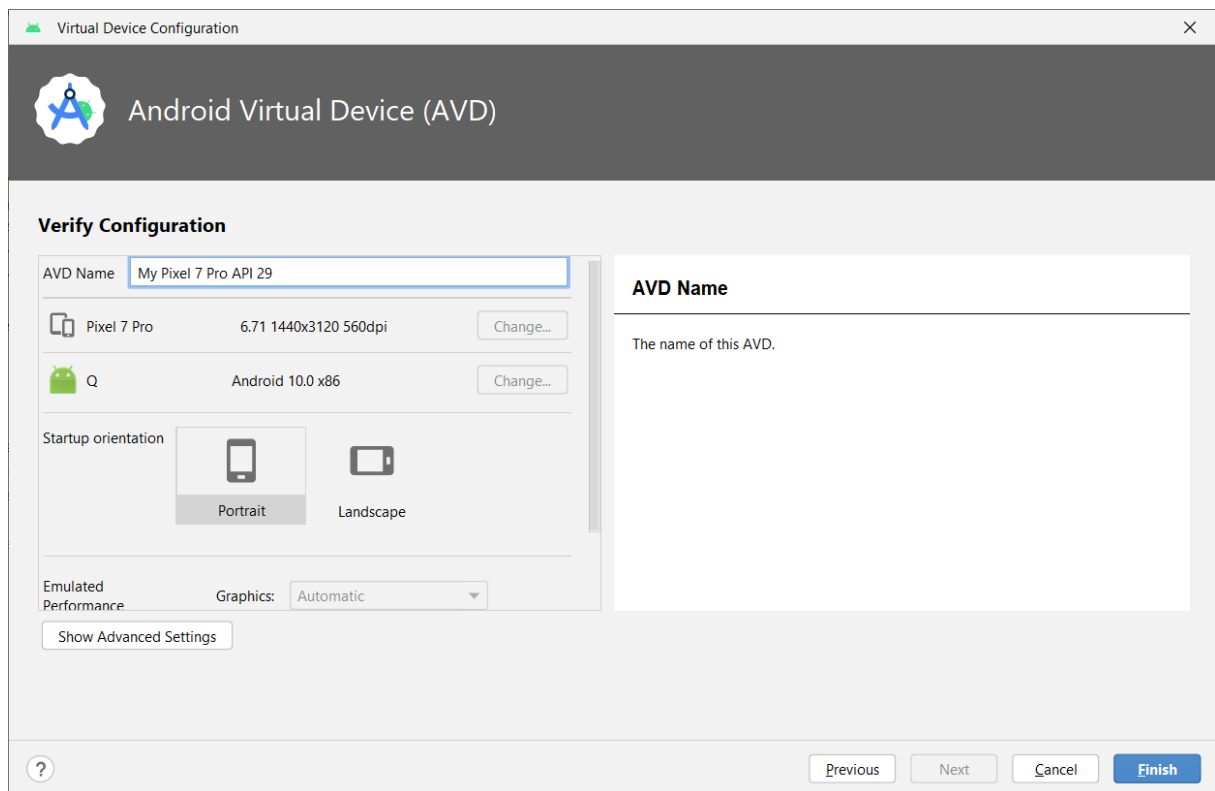
Tworząc nowe urządzenie wirtualne wybieramy jego kategorię (rodzaj) oraz konkretne urządzenie (nazwa, rozmiar ekranu, rozdzielczość itp.).



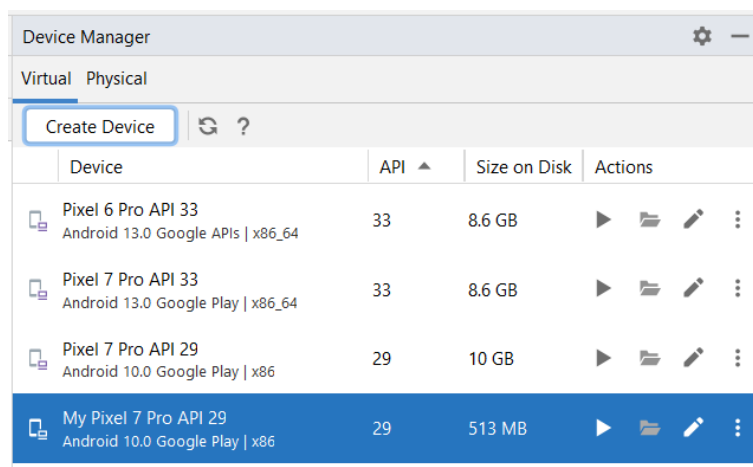
Po wybraniu wirtualnego urządzenia klikamy *Next* i wybieramy wersję systemu dla tego urządzenia.




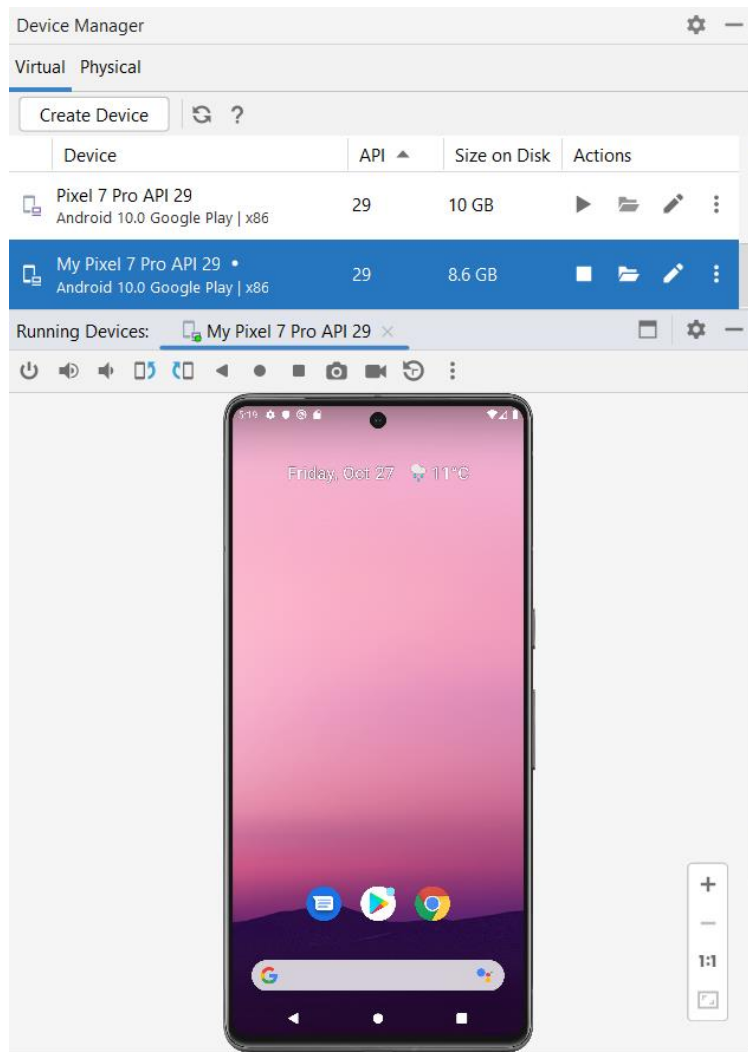
Dalej mamy okienko podsumowujące, w którym możemy zmienić domyślną nazwę urządzenia oraz zmienić samo urządzenie czy system.





Po dokonaniu wyboru klikamy *Finish*. W tym momencie na naszej liście urządzeń dostępne mamy nowe urządzenie.



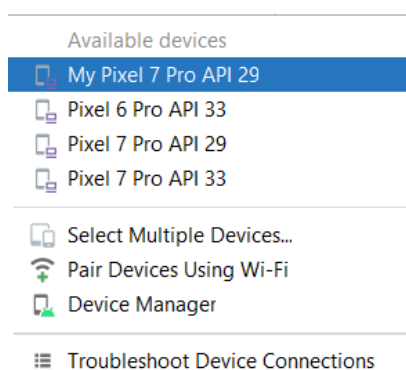
Teraz klikając na przycisk  obok nazwy wybranego urządzenia wirtualnego możemy je uruchomić.




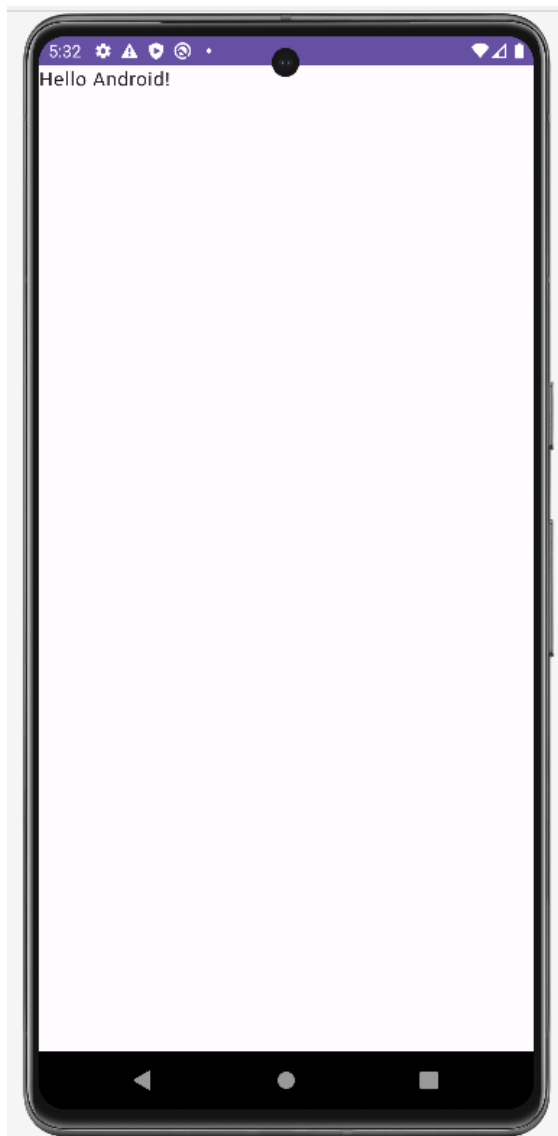
Przycisk  widoczny na powyższym rysunku pozwala na widok emulatora w osobnym oknie, co poprawia widoczność ekranu emulowanego urządzenia.

Chcąc zakończyć pracę emulatora klikamy przycisk  obok nazwy uruchomionego urządzenia wirtualnego.


Aby przypisać wybrany emulator urządzenia do naszego projektu ponownie wybieramy z menu głównego Android Studio **Run/Select Device...** i następnie wybieramy urządzenie.



Następnie klikając na przycisk , znajdujący się w górnej części okna Android Studio (lub klikając kombinację klawiszy *Shift + F10*), uruchamiamy naszą aplikację na wybranym emulatorze urządzenia.

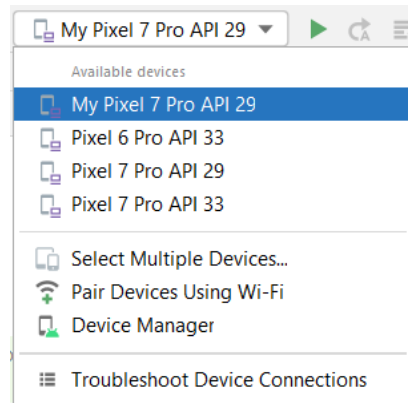


Jeżeli chcemy, aby przy uruchamianiu projektu na emulatorze nie było konieczne każdorazowe jego ponowne uruchamianie – pozostawiamy otwarte okienko emulatora, a jedynie kończymy działanie

naszej aplikacji na emulowanym urządzeniu (przycisk  w górnej części okna Android Studio lub kombinacja klawiszy *Ctrl + F2*).

Istnieje również inny sposób uruchamiania aplikacji na wybranym emulatorze. Wystarczy w górnej części okna Android Studio wybrać z listy rozwijanej żądane urządzenie i następnie kliknąć przycisk





ZADANIE

Utwórz nowy projekt w AndroidStudio (szablon *Empty Activity*) i uruchom go na wybranym emulatorze.


Uruchamianie aplikacji na urządzeniu rzeczywistym

Większość telefonów i tabletów z systemem Android może być podłączona do komputera za pomocą kabla USB. Domyślnie połączenie USB ustanowione między urządzeniem z Androidem i komputerem jest ograniczone tylko do transferu plików. Aby użyć swojego urządzenia do rozwijania aplikacji na Androida, musimy dokonać kilku zmian konfiguracyjnych.

W systemie Android wszystkie ustawienia przeznaczone dla deweloperów są domyślnie ukryte. Aby je wyświetlić, otwieramy *Ustawienia* na swoim urządzeniu i przechodzimy do ekranu *Informacje o telefonie*. Następnie szukamy pozycji *Numer kompilacji* i stukamy w nią siedem razy.

Wówczas na ekranie *System* w ustawieniach naszego urządzenia powinniśmy zobaczyć menu *Opcji programisty*. Należy je otworzyć i upewnić się, że opcja *Debugowanie USB* jest zaznaczona. Teraz nasze urządzenie może być wykorzystywane do rozwijania aplikacji.

Następnie podłączamy nasze urządzenie do komputera za pomocą kabla USB. W Android Studio w *Device Manager* w zakładce *Physical* powinniśmy zobaczyć nasze urządzenie. Również w górnej części okna Android Studio na liście rozwijanej, gdzie wcześniej wybieraliśmy emulator, powinniśmy zobaczyć nasze urządzenie jako domyślnie wybrane do uruchomienia aplikacji.

Teraz klikając przycisk  obok nazwy naszego urządzenia sprawimy, że aplikacja zostanie zainstalowana i uruchomiona na naszym rzeczywistym urządzeniu.

Modyfikacja projektu

Zmiana wyświetlanego tekstu

Spójrzmy teraz na kod zawarty w pliku *MainActivity.kt*. Zawiera on kilka automatycznie generowanych funkcji, w szczególności funkcje *onCreate()* i *setContent()*.

```

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MyTestAppTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Greeting( name: "Android")
                }
            }
        }
    }
}

```

Funkcja `onCreate()` jest funkcją uruchomieniową aplikacji Android i wywołuje inne funkcje w celu zbudowania interfejsu użytkownika.

Funkcja `setContent()` będąca w funkcji `onCreate()` służy do definiowania układu za pomocą funkcji komponowalnych. Wszystkie funkcje oznaczone adnotacją `@Composable` można wywołać z funkcji `setContent()` lub z innych funkcji `Composable`. Adnotacja informuje kompilator Kotlin, że ta funkcja jest używana przez *Jetpack Compose* do generowania interfejsu użytkownika.

Następnie spójrzmy na funkcję `Greeting()`. Jest ona funkcją komponowalną (*Composable*) pobierającą pewne dane wejściowe i generującą to, co jest widoczne na ekranie.

```

@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}

```

W porównaniu do standardowych wyróżnia się ona kilkoma rzeczami:

- Posiada adnotację `@Composable`
- Nazwa takiej funkcji rozpoczyna się od wielkiej litery
- Funkcja tego typu nie może nic zwracać

W tej chwili funkcja `Greeting()` pobiera imię (parametr `name`) i wyświetla napis *Hello* z dołączonym imieniem.

ZADANIE

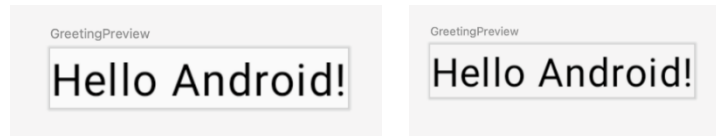
Zmodyfikuj kod funkcji `Greeting()` tak, aby wyświetlał tekst: „*Hi, my name is*” oraz imię podane jako parametr funkcji (podaj swoje imię).

Zauważ, że podczas modyfikacji kodu Android Studio automatycznie aktualizuje podgląd wyglądu aplikacji:

GreetingPreview

Hi, my name is Android!

Funkcja `GreetingPreview()` pozwala zobaczyć jak wygląda nasza aplikacja bez konieczności jej ponownego buildowania. Umożliwiają to adnotacje `@Composable` i `@Preview`. Adnotacja `@Preview` informuje Android Studio, że ten element kompozycji powinien być pokazany w widoku *Design*. Adnotacja `@Preview` przyjmuje parametr o nazwie `showBackground`, który ustawiony na wartość `true` dodaje tło do komponowanego podglądu. Ponieważ Android Studio domyślnie używa jasnego motywu edytora, może być trudno dostrzec różnicę między `showBackground = true` i `showBackground = false`.



ZADANIE

Zmodyfikuj kod funkcji `GreetingPreview()` wprowadzając swoje imię w miejsce parametru wywołania funkcji `Greeting()`.



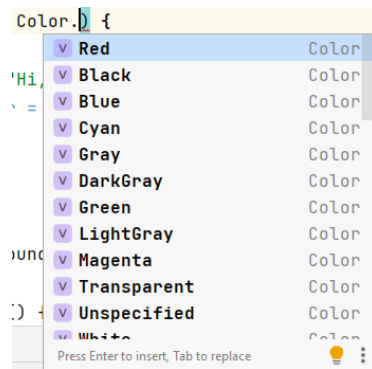
Zmiana koloru tła

Aby zmienić kolor tła elementu `Text`, musimy otoczyć go kontenerem `Surface`, który reprezentuje sekcję interfejsu użytkownika, gdzie można zmieniać wygląd np. kolor tła lub obramowanie.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface() {
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier
        )
    }
}
```

Następnie do kontenera `Surface` dodajemy parametr `color` i ustawiamy jego wartość na wybrany kolor z pakietu `androidx.compose.ui.graphics.Color` np. `Color.Red`.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface(color = Color.Red) {
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier
        )
    }
}
```



Wspomniany wyżej pakiet należy zaimportować do projektu.

```
import androidx.compose.ui.graphics.Color
```

Efekt zmian możemy dostrzec w widoku *Design*.



ZADANIE

Zmodyfikuj kod funkcji *Greeting()* ustawiając wybrany kolor tła pod tekstem.

Dodanie odstępów

Nasz komponent tekstowy posiada już tło, dodamy teraz odstępy (marginesy) wokół tekstu. Do tego celu wykorzystamy *Modifier*.

Modifier służy do ulepszania lub ozdabiania kompozycji. Jednym z modyfikatorów, których możemy użyć, jest modyfikator *padding*, który dodaje odstęp wokół elementu (w tym przypadku wokół tekstu). Osiąga się to za pomocą funkcji *Modifier.padding()*.

Każdy element komponowalny powinien mieć opcjonalny parametr typu *Modifier*. Powinien to być pierwszy opcjonalny parametr.

W naszym projekcie dodamy marginesy o rozmiarze 20 dp.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface(color = Color.Red) {
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier.padding(20.dp)
        )
    }
}
```

Potrzebujemy zaimportować odpowiednie pakiety

```
import androidx.compose.foundation.layout.padding
import androidx.compose.ui.unit.dp
```

ZADANIE

Zmodyfikuj kod funkcji *Greeting()* ustawiając odstępy wokół tekstu.

Pierwszy layout

Metody tworzenia layoutu

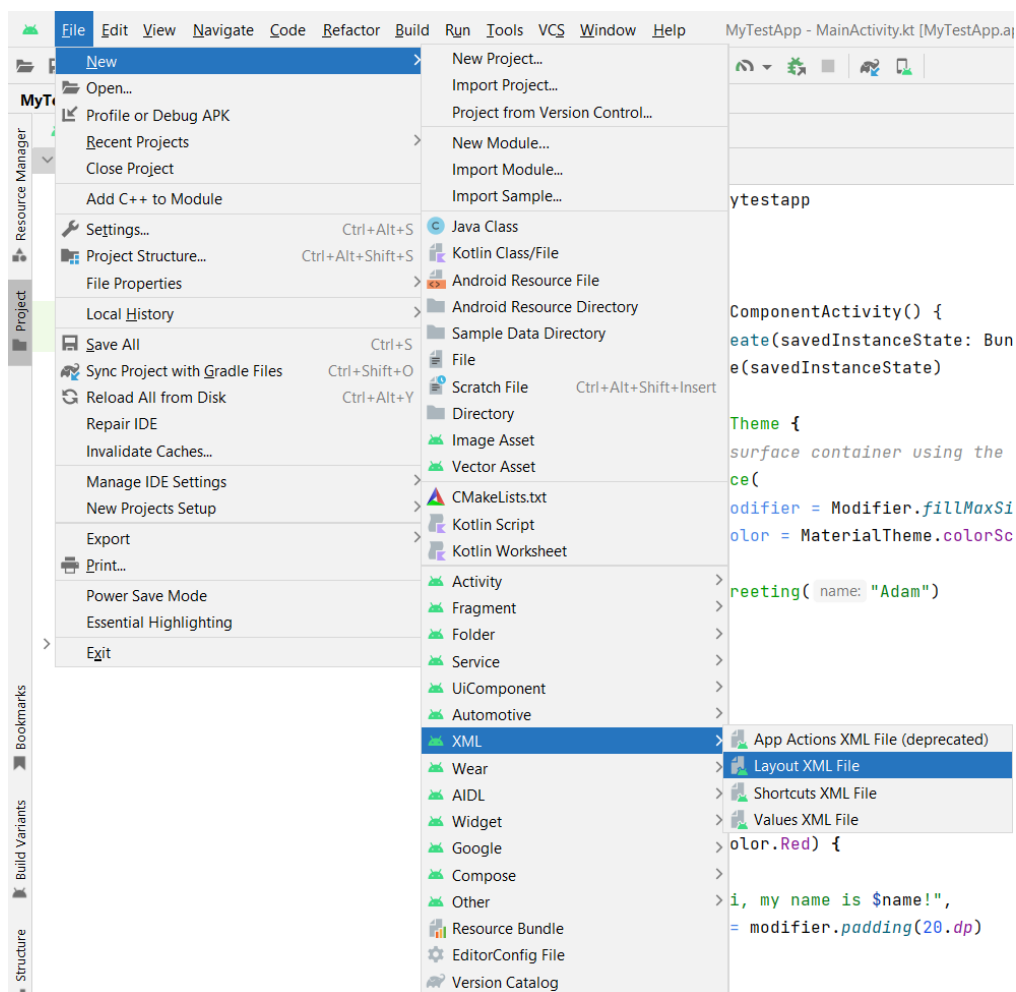
W Android Studio możemy tworzyć layout naszej aplikacji na 2 sposoby:

- poprzez ręczne pisanie kodu XML odpowiedzialnego za wygląd aplikacji
- poprzez przeciąganie wybranych kontroltek do widoku *Design* naszej aplikacji

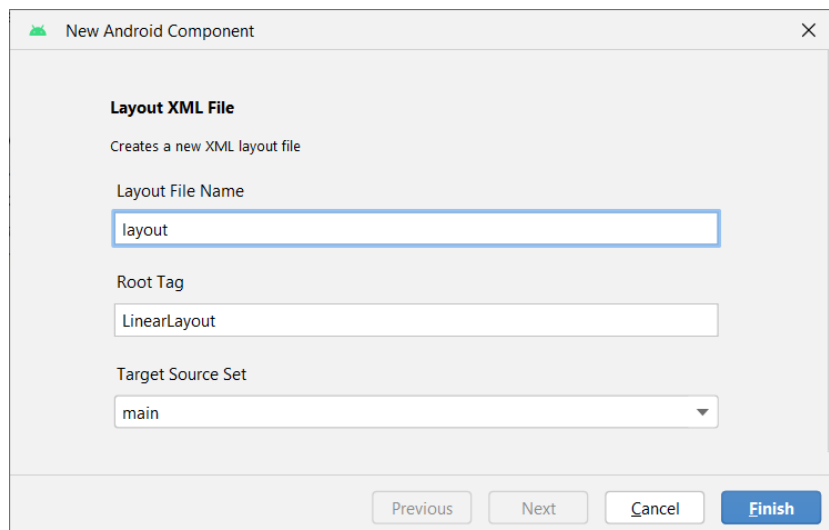
W naszym projekcie nie posiadamy obecnie katalogu *layout*, który jest domyślnym katalogiem plików odpowiedzialnych za wygląd aplikacji. Nie mamy także żadnego pliku XML z kodem określającym wygląd naszej aplikacji.

Aby utworzyć plik layoutu (wraz z katalogiem *layout*) należy z menu Android Studio wybrać kolejno:

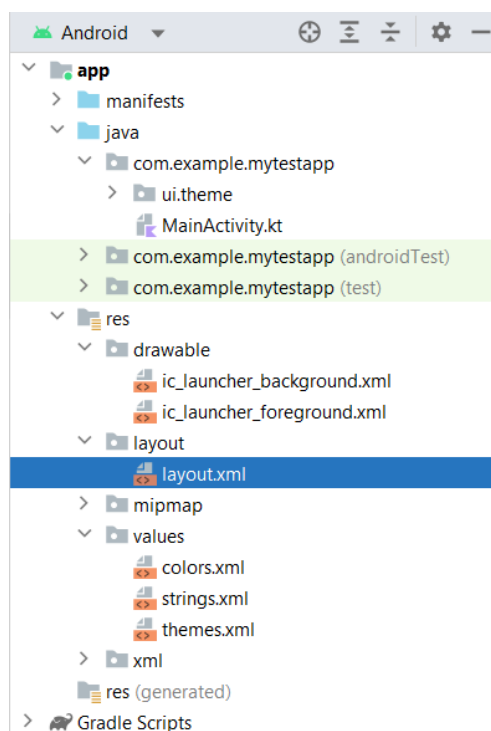
File/New/XML/Layout XML File



Następnie określamy m.in. nazwę pliku XML oraz główny znacznik (pozostawiamy domyślne):



W strukturze naszego projektu powinien pojawić się katalog *layout* wraz z plikiem XML odpowiedzialnym za wygląd naszej aplikacji.



ZADANIE

W utworzonym wcześniej projekcie utwórz plik layoutu.

„Kodowanie” layoutu


Plik konfiguracyjny layoutu naszej aplikacji zawiera kod XML. Odpowiada on m.in. za widoczne kontrolki, ich rozmiar, pozycję itp.

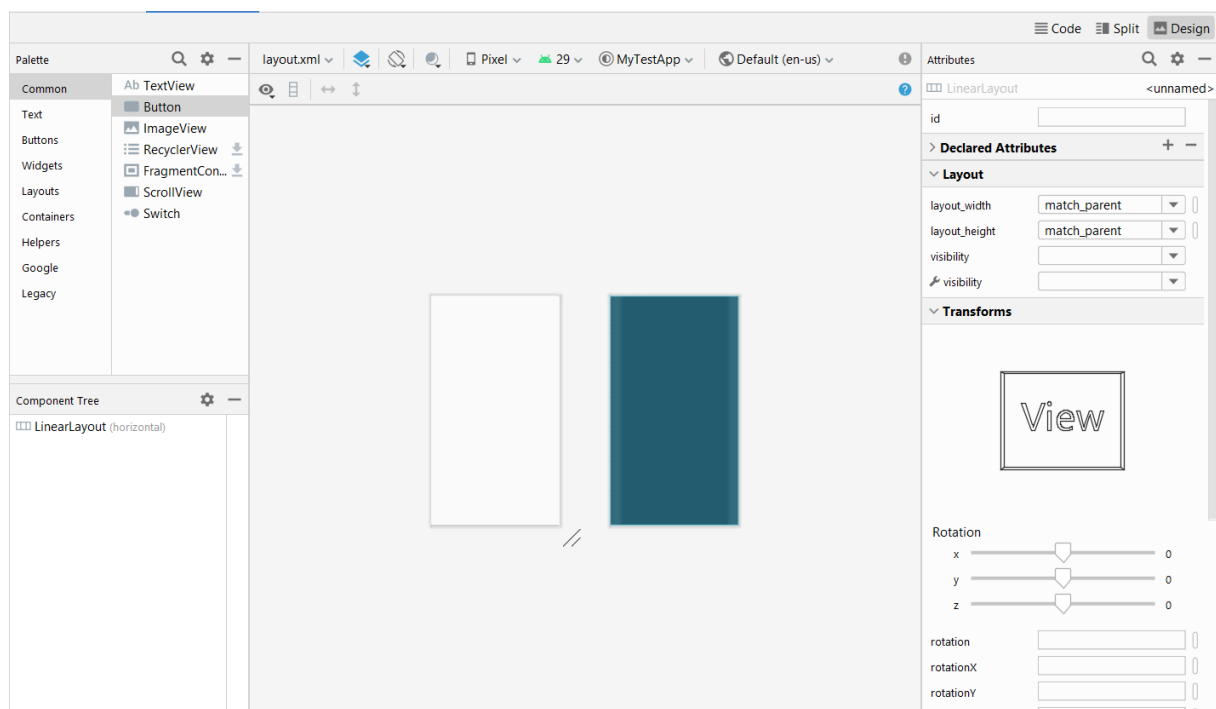

```
MainActivity.kt x layout.xml x
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent">
5
6 </LinearLayout>
7
```

Odpowiednio go modyfikując wpływamy na wyświetlaną zawartość aplikacji.

„Wyklikanie” layoutu

Innym sposobem na tworzenie aplikacji jest skorzystanie z designera naszej aplikacji. Aby go

wywołać, mając otwarty plik XML layoutu klikamy na zakładkę  Design w górnej części okna.



Po lewej stronie mamy dostępną paletę kontroltek pogrupowanych w kategorie oraz drzewo użytych komponentów. Po prawej z kolei znajdują się właściwości wybranej kontrolki.

Tworzenie layoutu

Linear Layout

Jednym z przykładów layoutu jest *LinearLayout*, który pozycjonuje komponenty aplikacji „od lewej”.

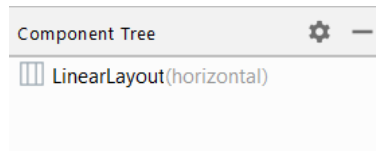
Spójrzmy teraz na kod utworzonego wcześniej pliku layoutu:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent">
5
6 </LinearLayout>
7
```

W linii 2 mamy znacznik otwierający *LinearLayout*, w którym ustawione są atrybuty:

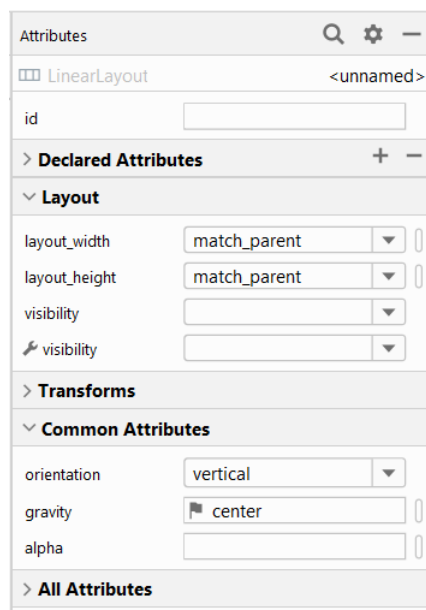
- szerokość (*layout_width*) na *match_parent* („dopasuj do rodzica”)
- wysokość (*layout_height*) na *match_parent* („dopasuj do rodzica”)

Po przejściu do zakładki *Design*, w drzewie komponentów widnieje nasz *LinearLayout*.



ZADANIE

W okienku atrybutów ustaw opcje zgodnie z poniższym rysunkiem.

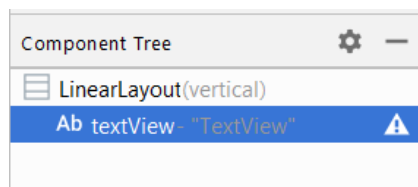


Ustawienie orientacji powoduje, że dodawane kontrolki będą umieszczane jedna pod drugą.
Ustawienie grawitacji z kolei powoduje „ściąganie” kontroltek do środka layoutu.

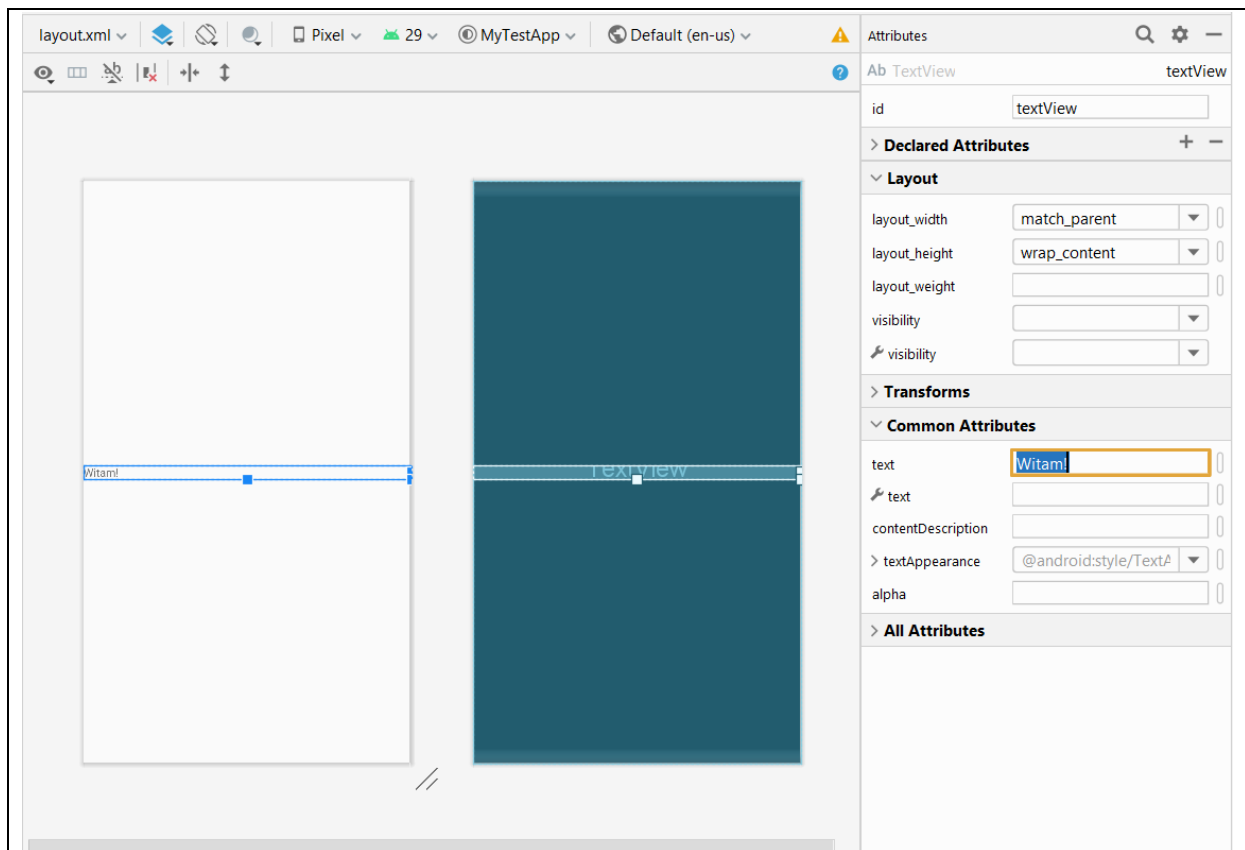
[Dodawanie kontroltek do drzewa komponentów](#)

ZADANIE

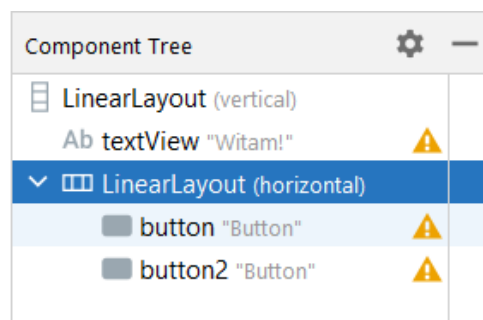
Dalej do naszego layoutu dodaj pole tekstowe *TextView*.



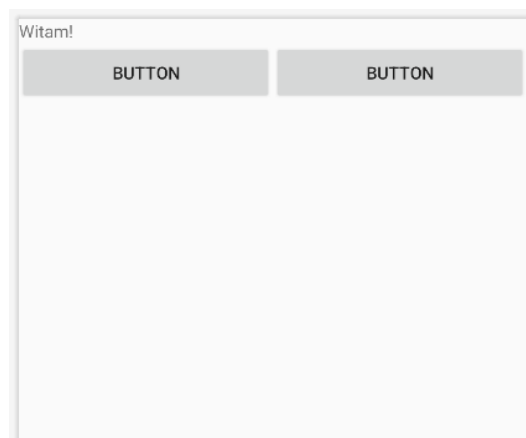
Ustaw tekst dla naszego komponentu („Witam!”).



Następnie dodaj kolejne kontrolki do drzewa komponentów, zgodnie z poniższym rysunkiem.



W tej chwili nasza aplikacja wygląda następująco:



Kod pliku XML z definicją layoutu uległ zmianie. Wygenerowane zostały kolejne linie kodu zgodnie z czynnościami wykonanymi w Designerze.

```
1      <?xml version="1.0" encoding="utf-8"?>
2      <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3          android:layout_width="match_parent"
4          android:layout_height="match_parent"
5          android:gravity="center"
6          android:orientation="vertical">
7
8          <TextView
9              android:id="@+id/textView"
10             android:layout_width="match_parent"
11             android:layout_height="wrap_content"
12             android:text="Witam!" />
13
14         <LinearLayout
15             android:layout_width="match_parent"
16             android:layout_height="match_parent"
17             android:orientation="horizontal">
18
19             <Button
20                 android:id="@+id/button"
21                 android:layout_width="wrap_content"
22                 android:layout_height="wrap_content"
23                 android:layout_weight="1"
24                 android:text="Button" />
25
26             <Button
27                 android:id="@+id/button2"
28                 android:layout_width="wrap_content"
29                 android:layout_height="wrap_content"
30                 android:layout_weight="1"
31                 android:text="Button" />
32         </LinearLayout>
33     </LinearLayout>
```

Ustawienie parametru np. dla przycisku (znacznik *Button*) *layout_width="wrap_content"* oznacza „dopasuj do zawartości”.


ZADANIE

Zmodyfikuj kod layoutu do następującej postaci:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:gravity="center"
6      android:orientation="vertical">
7
8      <TextView
9          android:id="@+id/textView"
10         android:layout_width="match_parent"
11         android:layout_height="wrap_content"
12         android:text="Witam!"
13         android:padding="24dp"/>
14
15     <LinearLayout
16         android:layout_width="match_parent"
17         android:layout_height="match_parent"
18         android:orientation="horizontal">
19
20         <Button
21             android:id="@+id/button"
22             android:layout_width="wrap_content"
23             android:layout_height="wrap_content"
24             android:layout_weight="1"
25             android:text="Cześć!" />
26
27         <Button
28             android:id="@+id/button2"
29             android:layout_width="wrap_content"
30             android:layout_height="wrap_content"
31             android:layout_weight="1"
32             android:text="Dzień dobry!" />
33     </LinearLayout>
34 </LinearLayout>
```

W linii 13 użyty został margines wewnętrzny dla *TextView* o szerokości *24dp* (dp – jednostka niezależna od gęstości pikseli). Zmieniony został również tekst wyświetlany na przyciskach *Button*.

Ostrzeżenia

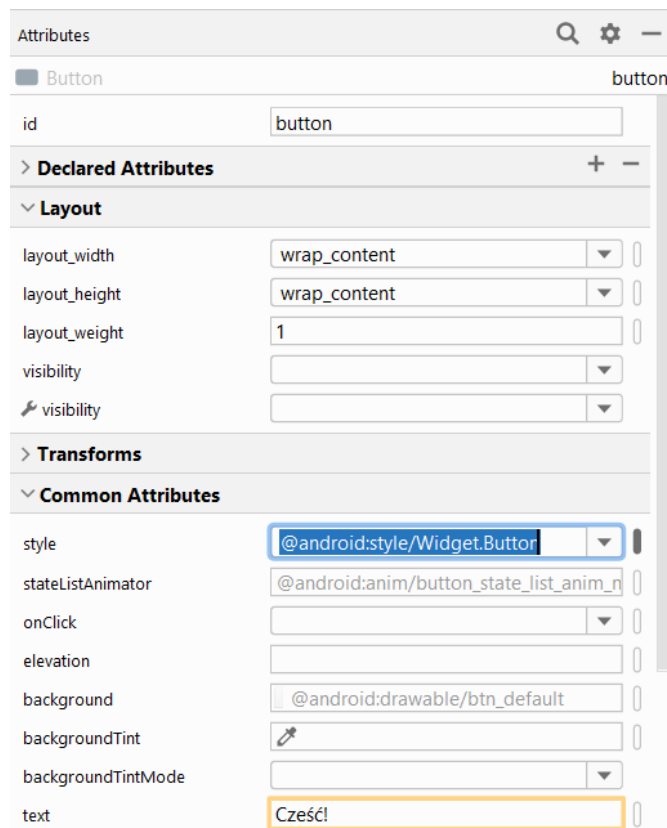
W naszym projekcie pojawiły się pewne ostrzeżenia. Aby je zobaczyć klikamy na ikonę  w górnej części widoku *Design*.

- layout.xml C:\Users\Adam\AndroidStudioProjects\MyTestApp\app\src\main\res\layout 5 problems
 - button <Button>: Button should be borderless
 - button <Button>: Hardcoded text
 - button2 <Button>: Button should be borderless
 - button2 <Button>: Hardcoded text
 - textView <TextView>: Hardcoded text
- Layout Validation 2 problems
 - The button button <Button> is too wide
 - The button button2 <Button> is too wide

Ostrzeżenia w liniach 1 i 3 dotyczą stylów przycisków.

ZADANIE

Zmień style przycisków w następujący sposób:



Pozostałe 3 ostrzeżenia (plik *layout.xml*) dotyczą tekstów ustawionych na naszych komponentach. Zostały one niejako „wyrte” na nich. Nie jest to dobre rozwiązanie, gdyż zmieniając język naszej aplikacji (np. chcąc uzyskać wersję anglojęzyczną) musimy dokonywać ręcznie zmian napisów dla każdej z kontrolki, co przy rozbudowanej aplikacji może być (i jest) uciążliwe. Na pomoc przychodzi plik *strings.xml* w katalogu *values*.

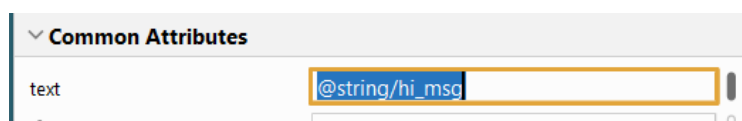
[Plik strings.xml](#)

ZADANIE

Zmodyfikuj plik *strings.xml* dodając 3 linie kodu.

```
1 <resources>
2     <string name="app_name">MyTestApp</string>
3     <string name="left_btn">Cześć!</string>
4     <string name="right_btn">Dzień dobry!</string>
5     <string name="hi_msg">Witam!</string>
6 </resources>
```

Następnie zmień atrybut *text* dla *TextView* w następujący sposób (w *layout.xml*):



Podobnie zmodyfikuj ustawienia tekstu na obydwu przyciskach.

Po dokonanych zmianach ostrzeżenia zniknęły, a dzięki zastosowaniu odwołań kontroltek do pliku *strings.xml* możemy teraz w łatwy sposób zmieniać tekst kontroltek modyfikując tenże plik.

Identyfikatory komponentów

Każdy z komponentów naszej aplikacji posiada (powinien posiadać) swój unikalny identyfikator, dzięki któremu w łatwy sposób możemy odwołać się do odpowiedniej kontrolki. Nazwy identyfikatorów powinny jednoznacznie wskazywać o którą kontrolkę nam chodzi. Jednym ze sposobów nadawania nazw identyfikatorów jest użycie jako id nazwy komponentu i jego zadania, np. *button_witam*. Dla nazw złożonych możemy użyć znaku podkreślenia lub tzw. Camel Case np. *buton_dzieńDobry*

ZADANIE

Zmień identyfikatory wszystkich komponentów na bardziej zrozumiałe.

onClick i zdarzenia przycisku

W tej sekcji poznamy jak obsługiwać zdarzenia w naszej aplikacji, np. kliknięcie na przycisk.

ZADANIE

Zmodyfikuj zawartość klasy głównej w następujący sposób:

```
1 package com.example.mytestapp
2
3 import ...
4
5
6
7
8 class MainActivity : AppCompatActivity() {
9     override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11
12         setContentView(R.layout.layout)
13
14         findViewById<Button>(R.id.button_czesc).setOnClickListener { it: View!
15             findViewById<TextView>(R.id.textView_powitanie).setText("Cześć!")
16         }
17     }
18 }
19 }
```

W liniach 14-15 dodaliśmy obsługę kliknięcia na przycisk o id=`button_czesc`. W wyniku tego kliknięcia zmieni się tekst wyświetlany na komponencie `TextView` (id=`textView_powitanie`).

ZADANIE

Uruchom aplikację i przetestuj działanie przycisku. W podobny sposób obsłuż zdarzenie kliknięcia na drugi przycisk.

Wyświetlanie tekstu za pomocą elementu Toast

Tekst możemy wyświetlać nie tylko w `TextView`, ale np. za pomocą wyskakującej belki `Toast`.

ZADANIE

Zmodyfikuj kod klasy głównej odpowiedzialny za akcję przycisku `buton_czesc` w następujący sposób:

```
15         findViewById<Button>(R.id.button_czesc).setOnClickListener { it: View!
16             findViewById<TextView>(R.id.textView_powitanie).setText("Cześć!")
17             var message = Toast.makeText(applicationContext, "Cześć!", Toast.LENGTH_LONG)
18             message.show()
19         }
```

Przetestuj działanie aplikacji.

W linii 17 zdefiniowaliśmy zmienną `message`, która przechowuje powiadomienie `Toast`. Parametr `applicationContext` oznacza, że powiadomienie będzie wyświetlane w naszej aplikacji, zaś `Toast.LENGTH_LONG` oznacza, że będzie ono widniało na ekranie przez dłuższy czas (można zmienić na `LENGTH_SHORT`).

Samo utworzenie Toasta to za mało, trzeba go jeszcze wyświetlić. Dlatego w linii 18 wywołujemy metodę `show()` na naszej zmiennej `message`.

ZADANIE

Dokonaj modyfikacji kodu klasy głównej wg poniższego rysunku.

```
9  class MainActivity : ComponentActivity() {
10      override fun onCreate(savedInstanceState: Bundle?) {
11          super.onCreate(savedInstanceState)
12
13          setContentView(R.layout.layout)
14
15          findViewById<Button>(R.id.button_czesc).setOnClickListener { it: View!
16              findViewById<TextView>(R.id.textView_powitanie).setText("Cześć!")
17              var message = Toast.makeText(applicationContext, text: "Cześć!", Toast.LENGTH_LONG)
18              message.show()
19          }
20
21          findViewById<Button>(R.id.button_dzienDobry).setOnClickListener { it: View!
22              findViewById<TextView>(R.id.textView_powitanie).setText("Dzień dobry!")
23              var message = Toast.makeText(applicationContext, text: "Dzień dobry!", Toast.LENGTH_LONG)
24              message.show()
25          }
26      }
27  }
28  }
```

Tym razem w linii 23 utworzony został Toast, na którym widniał będzie tekst pobrany z przycisku *button_dzienDobry*.

Poniżej kilka zrzutów ekranu z działania naszej aplikacji.



ZADANIE

Utwórz nową aplikację, która będzie realizować zadanie kalkulatora z obsługą 4 działań: dodawania, odejmowania, mnożenia i dzielenia. Do wprowadzania danych użyj przycisków z cyframi i działaniami. Treść działania wyświetl za pomocą *TextView*. Po kliknięciu przycisku *Oblicz* zawartość *TextView* zamień na wynik obliczeń. Jeżeli wystąpi dzielenie przez 0 wyświetl *Toast* ze stosownym komunikatem.