

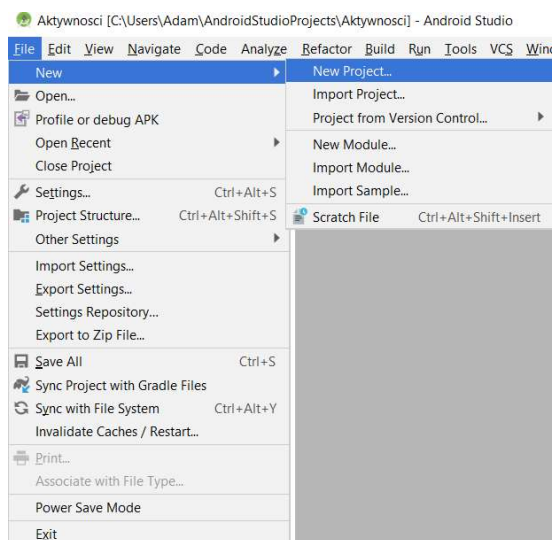
Kotlin & Android w AndroidStudio (3.5.1)

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.

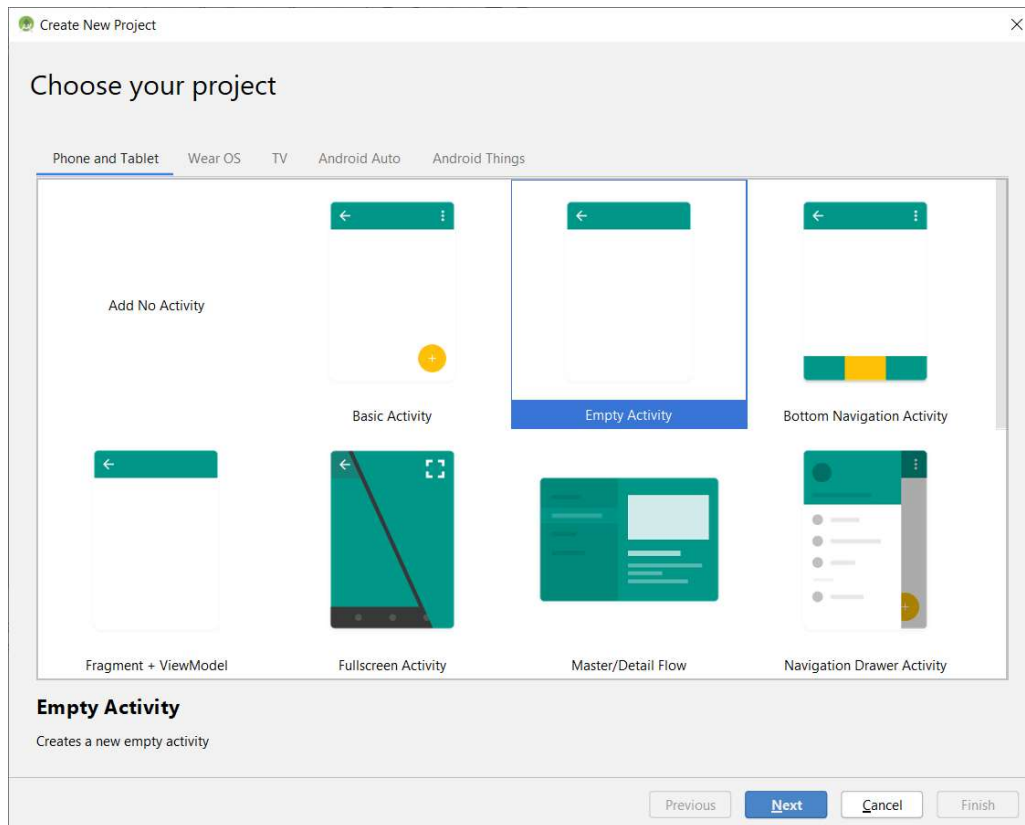
Tworzenie nowego projektu

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

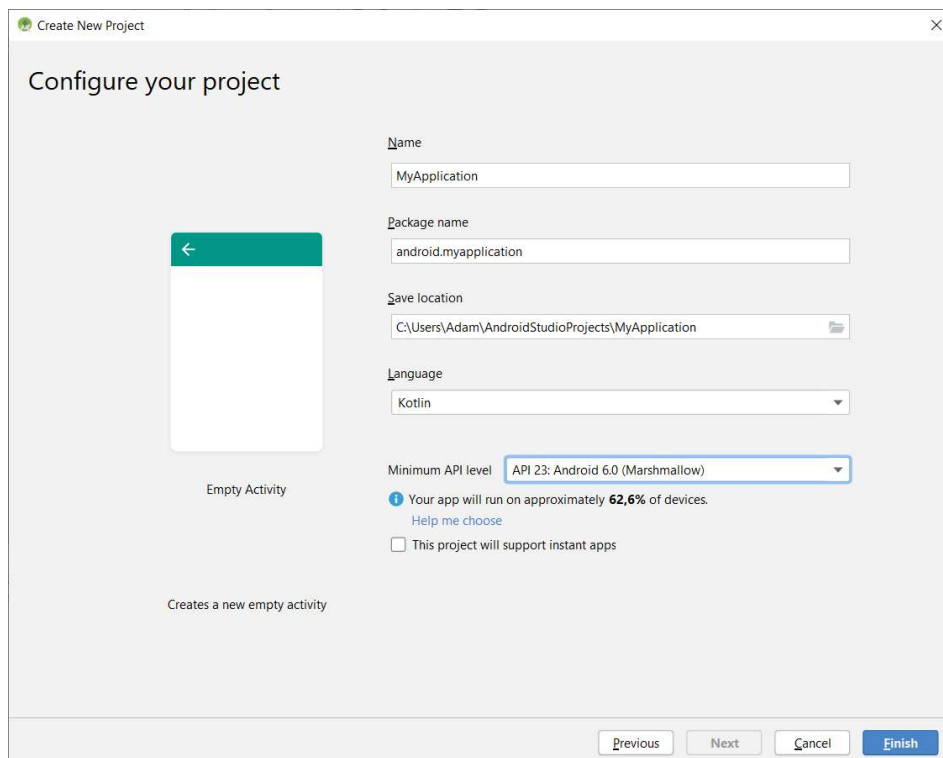


Nowa aktywność

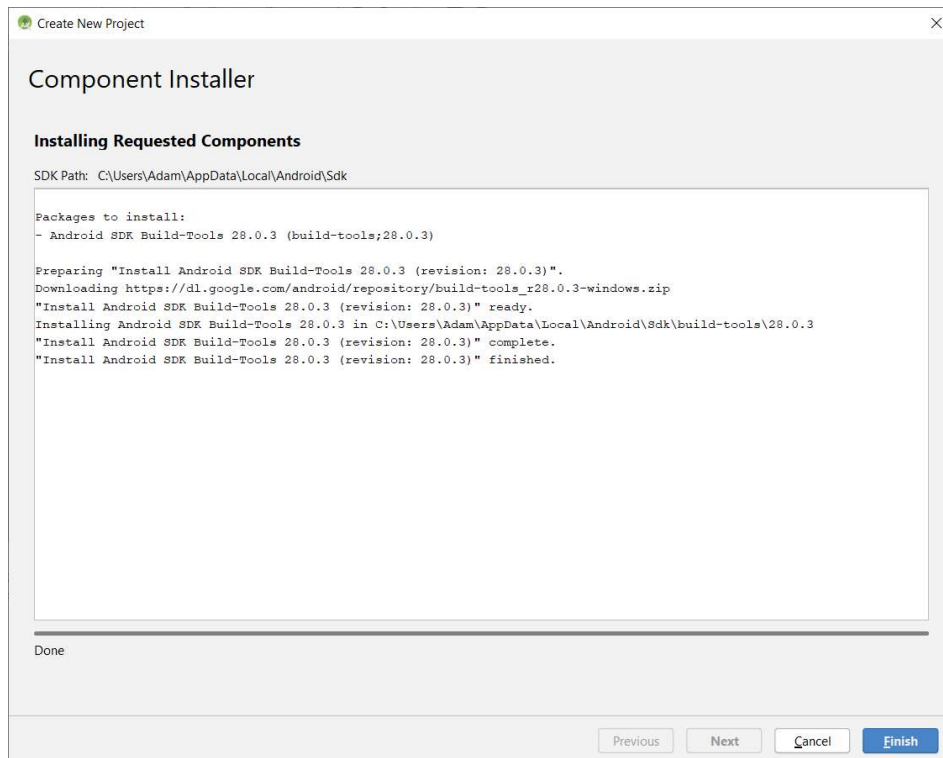
Następnie zostajemy poproszeni o wybór rodzaju aktywności. Dla celów tego laboratorium wybieramy **Empty Activity**.



Dalej konfigurujemy nasz projekt. Wprowadzamy nazwę aplikacji, nazwę pakietu (pozostawiamy bez zmian), wybieramy lokalizację, język programowania (zaznaczamy Kotlin!) oraz API dla naszej aplikacji (Android 6.0).



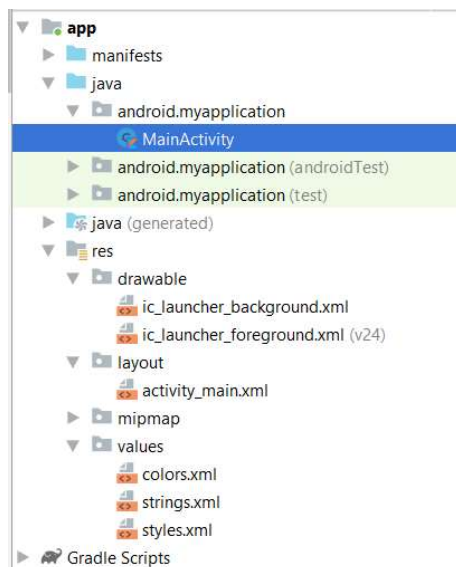
AndroidStudio instaluje potrzebne pakiety:



Klikamy **Finish**.

Struktura projektu

Projekt aplikacji AndroidStudio zawiera wiele pakietów i plików. Nas najbardziej interesują pakiety **java** i **res**. W pierwszym z nich znajdziemy plik głównej klasy projektu, zaś w drugim mamy dostęp m.in. do grafik, layoutu, pliki konfiguracyjne (katalog values).



Poza wspomnianymi pakietami mamy także skrypty Gradle'a.

Klasa MainActivity

Klasa **MainActivity** jest główną klasą projektu. Tutaj będziemy pisać kod odpowiedzialny za całą logikę aplikacji.

```

package android.myapplication

import ...

class MainActivity : AppCompatActivity() {

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

```

W powyższym kodzie mamy klasę **MainActivity** dziedziczącą po klasie **AppCompatActivity**, w której znajduje się funkcja **onCreate()** (przeciążona).

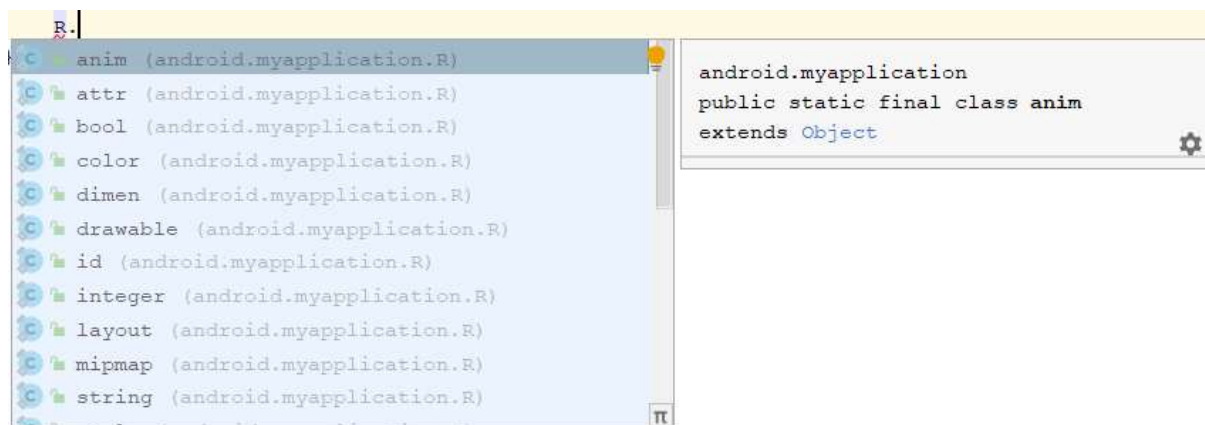
Dziedziczenie po **AppCompatActivity** oznacza kompatybilność wsteczną naszej aplikacji. Jeżeli nie chcemy z niej korzystać, to nasza klasa główna będzie dziedziczyć po **Activity**.

Linijka:

```
setContentView(R.layout.activity_main)
```

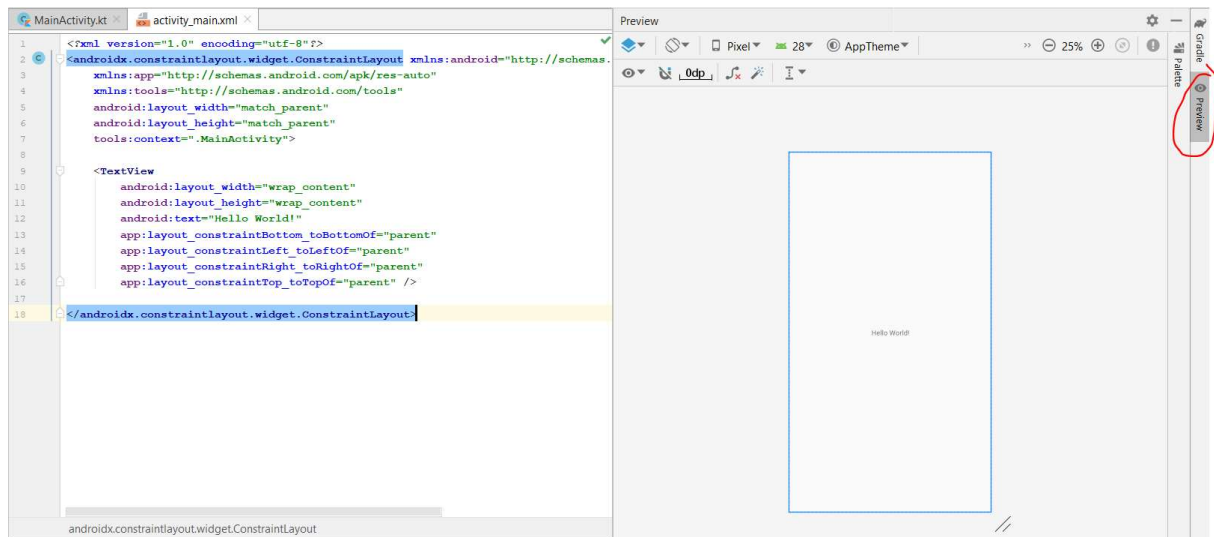
ustawia nam widok aplikacji na layout, który został zdefiniowany w activity_main.

R jest automatycznie generowaną klasą zawierającą wszystkie nasze zasoby (m.in. layouty, style, identyfikatory):

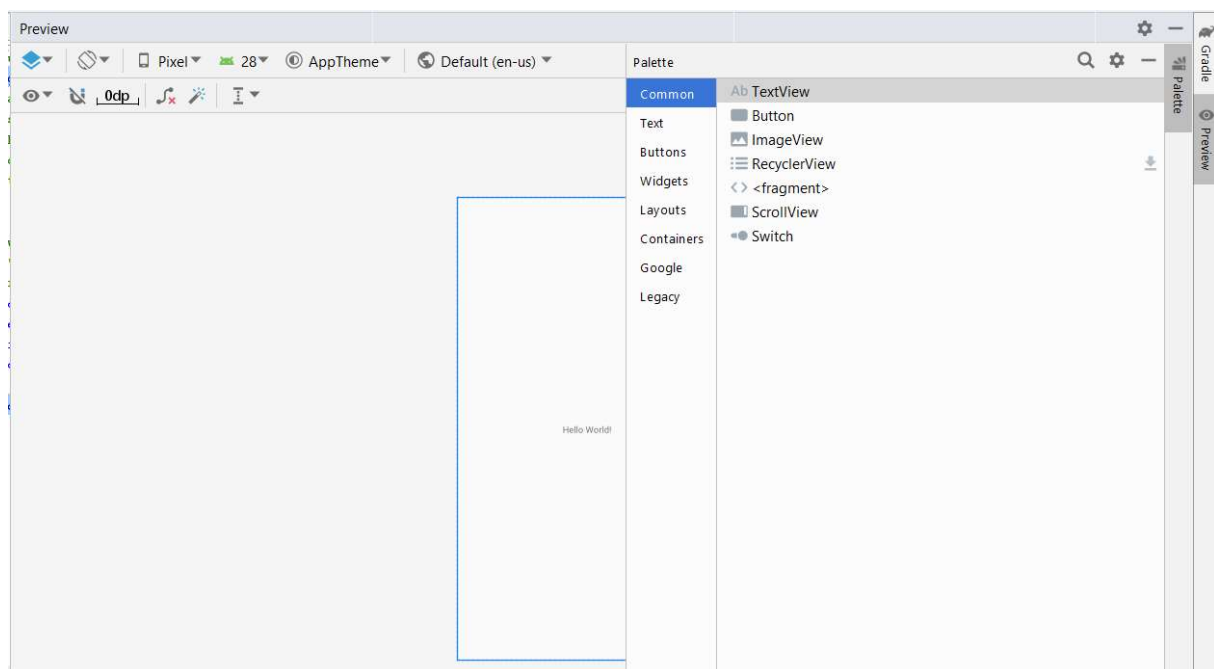


Widok layoutu

W folderze **res/layout** możemy zobaczyć jak wygląda nasza aplikacja. Klikamy na plik **activity_main.xml** oraz na zakładkę **Preview**.

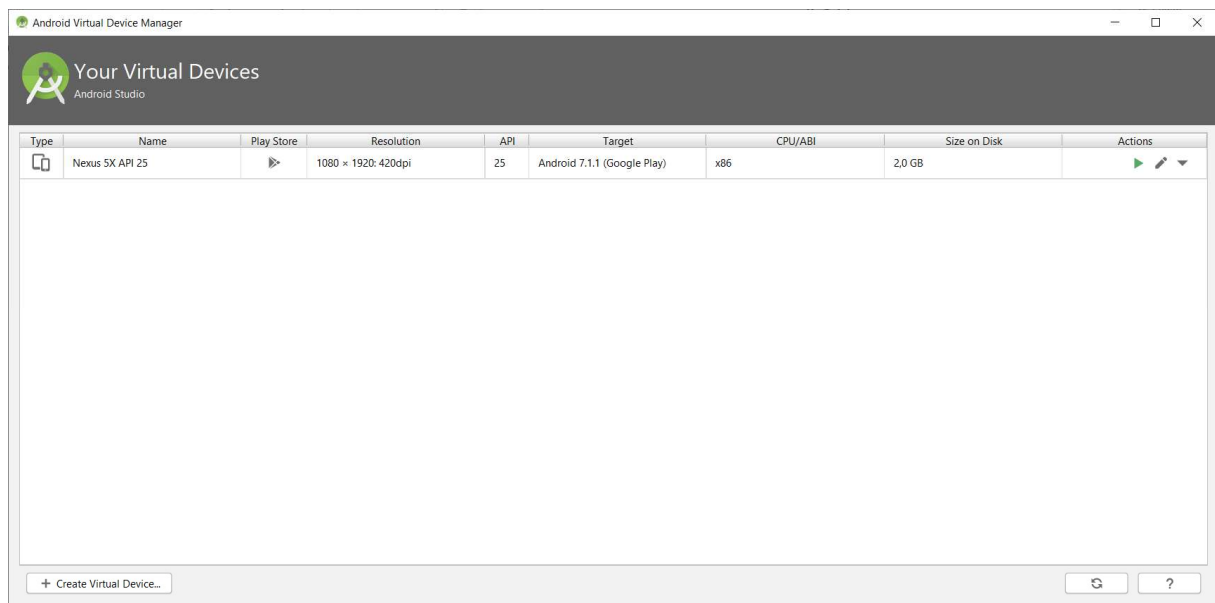


Ponadto w widoku layoutu możemy wywołać paletę kontroltek do tworzenia naszej aplikacji.

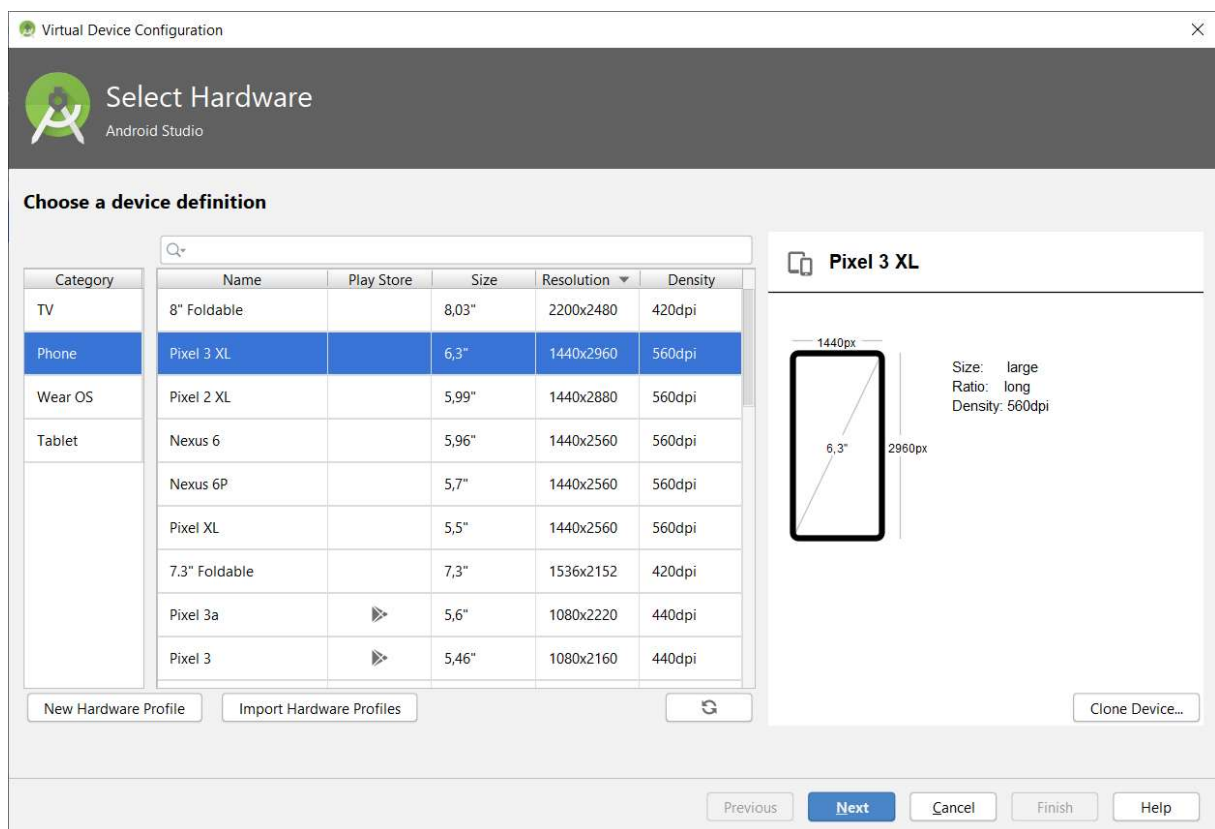


Uruchomienie aplikacji

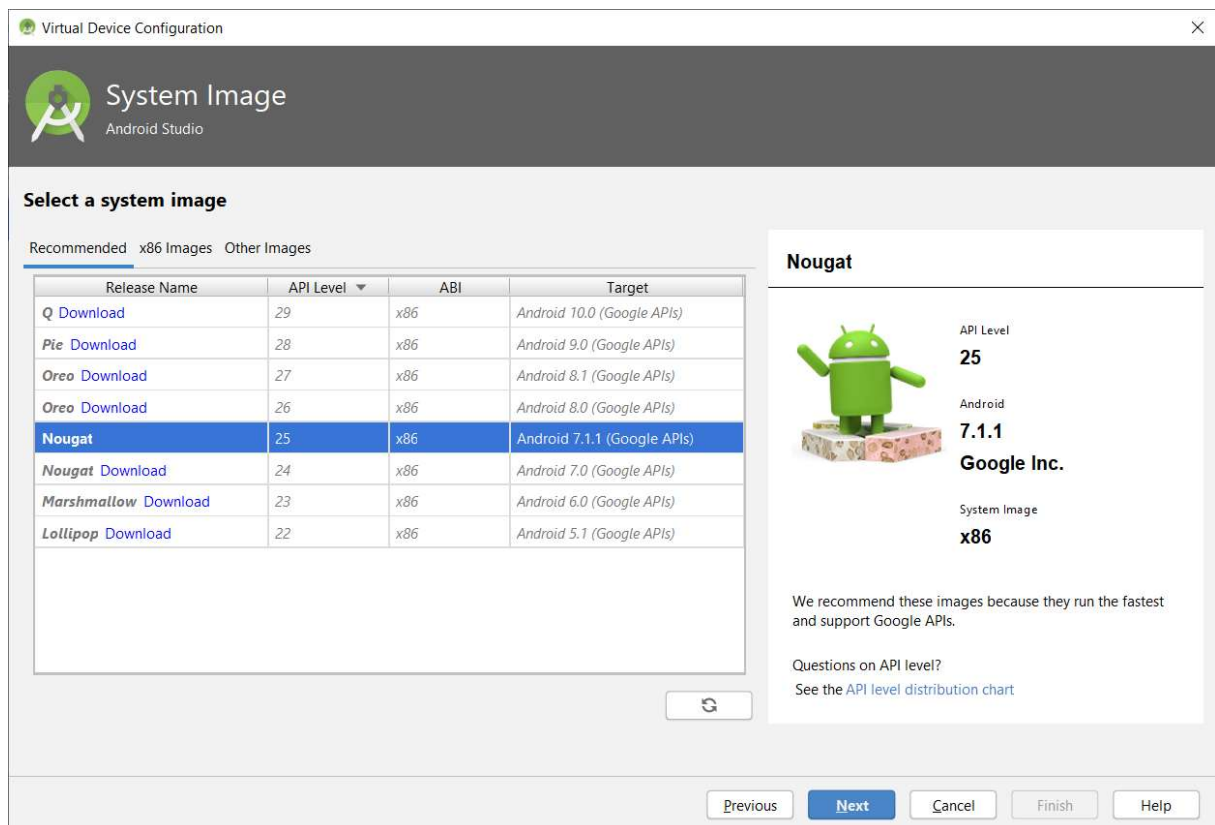
Aby uruchomić aplikację skorzystamy z emulatora. W tym celu wybieramy **Run/Select Device...**, a następnie **Open AVD Manager**. Wywołany zostanie manager urządzeń wirtualnych.



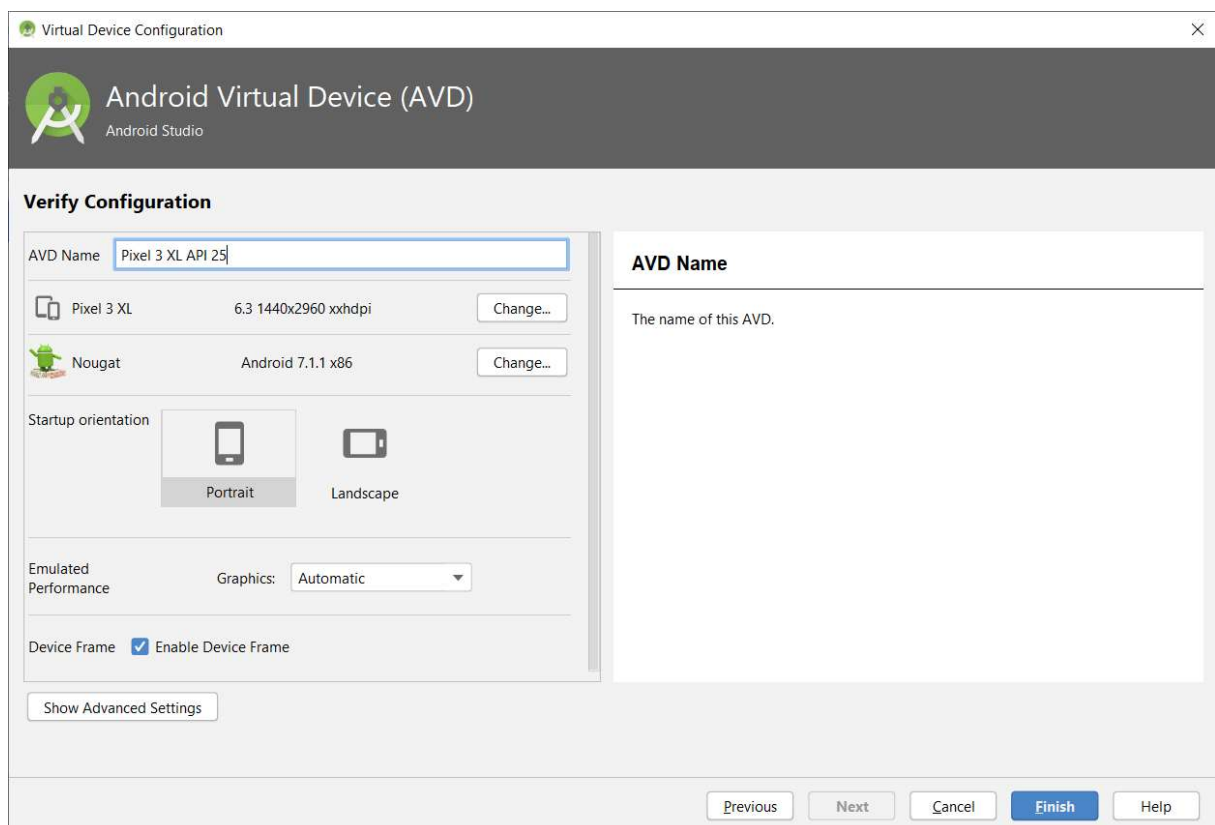
Mamy tutaj listę zainstalowanych emulatorów oraz możemy dodać kolejne klikając na **Create Virtual Device**.



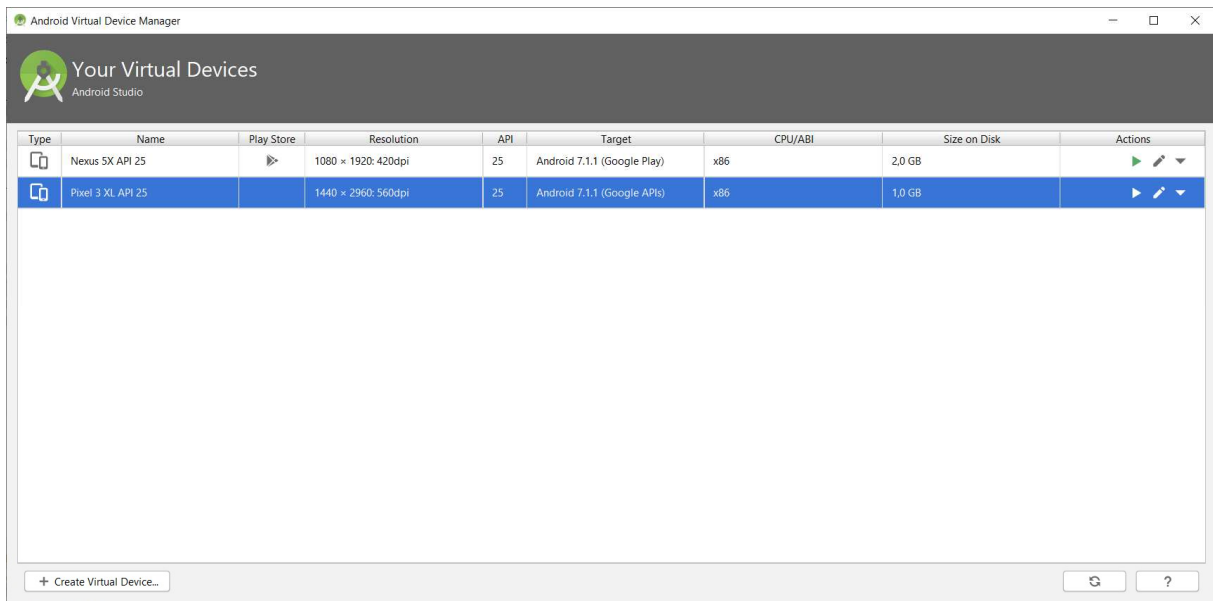
Po wybraniu wirtualnego urządzenia klikamy **Next** i wybieramy wersję systemu dla tego urządzenia.



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




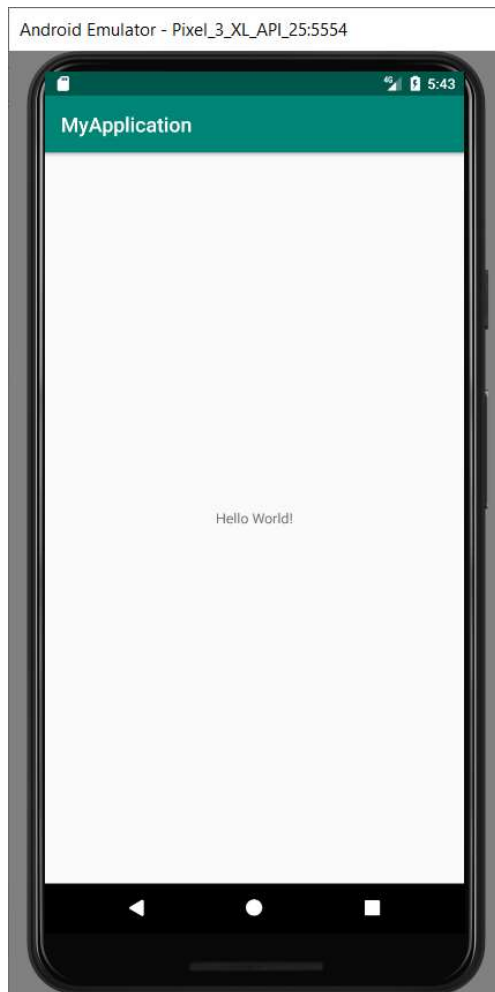
Po dokonaniu wyboru klikamy **Finish**. W tym momencie na naszej liście urządzeń dostępne mamy nowe urządzenie.



Aby przypisać wybrany emulator urządzenia do naszego projektu ponownie klikamy **Run/Select Device...** i następnie wybieramy urządzenie.



Teraz klikając na przycisk  w górnym menu 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.

ZADANIE

Utwórz nowy projekt w AndroidStudio i uruchom go.

Pierwszy layout

Metody tworzenia layoutu

W AndroidStudio 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

„Kodowanie” layoutu

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

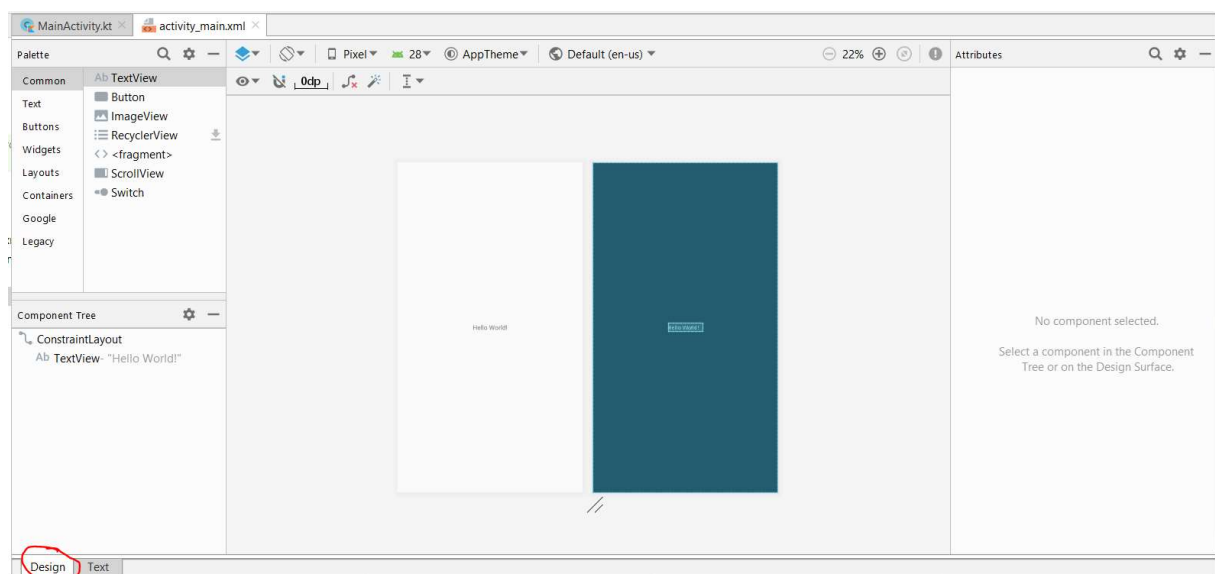


```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity">
8
9     <TextView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Hello World!"
13         app:layout_constraintBottom_toBottomOf="parent"
14         app:layout_constraintLeft_toLeftOf="parent"
15         app:layout_constraintRight_toRightOf="parent"
16         app:layout_constraintTop_toTopOf="parent" />
17
18 </androidx.constraintlayout.widget.ConstraintLayout>
```

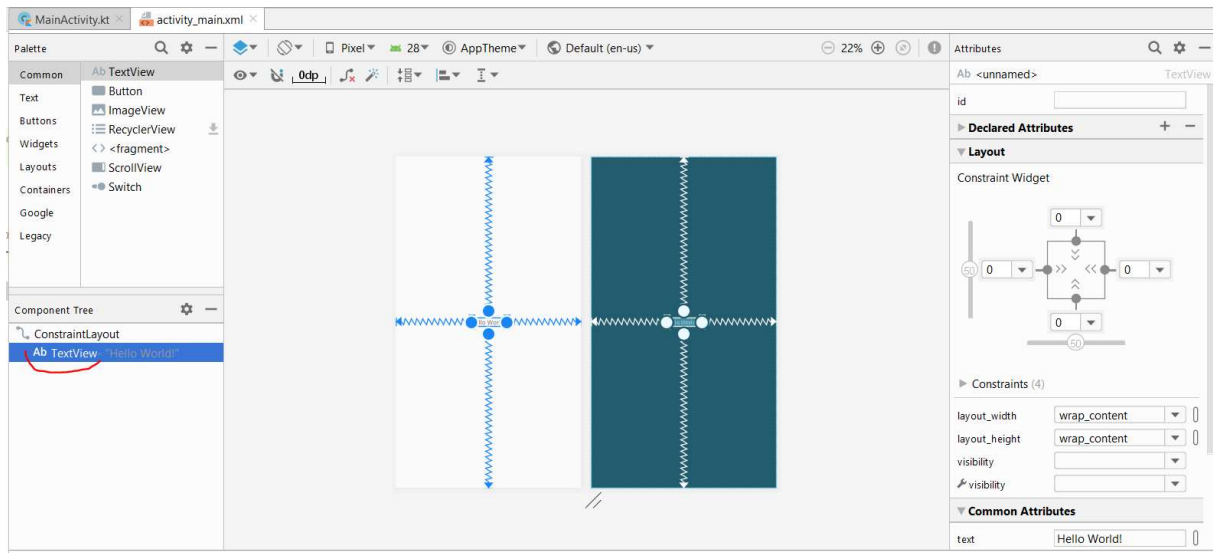
Odpowiednio go modyfikując wpływamy na wyświetlaną zawartość aplikacji.

„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 dolnej 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 **Linear Layout**, który pozycjonuje komponenty aplikacji „od lewej”.

ZADANIE

W utworzonym wcześniej projekcie przejdź do pliku layoutu i usuń jego zawartość pozostawiając tylko pierwszą linię kodu. Następnie dopisz poniższy fragment.

```

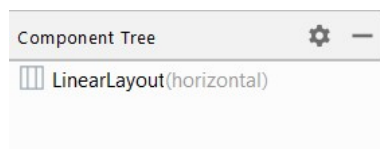
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent">
5 </LinearLayout>

```

W linii 2 mamy znacznik otwierający LinearLayout, w którym ustawiliśmy atrybuty:

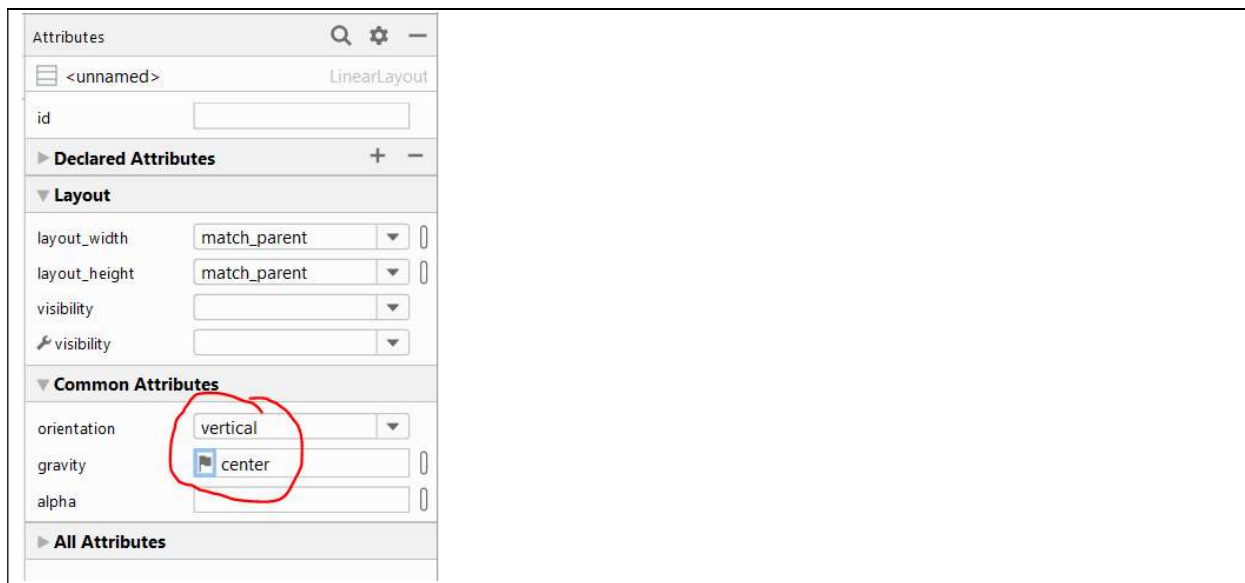
- szerokość (layout_width) na match_parent („dopasuj do rodzica”)
- wysokość (layout_height) na match_parent („dopasuj do rodzica”)

Po zapisaniu zmian w pliku i przejściu do zakładki **Design**, w drzewie komponentów widnieje nasz LinearLayout.



ZADANIE

W okienku atrybutów ustaw opcje zgodnie z poniższym rysunkiem.

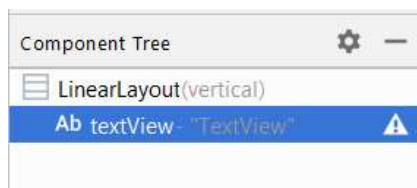


Ustawienie orientacji powoduje, że dodawane kontrolki będą umieszczane jedna pod drugą.
Ustawienie grawitacji z kolei powoduje „ściągnięcie” kontrolki do środka layoutu.

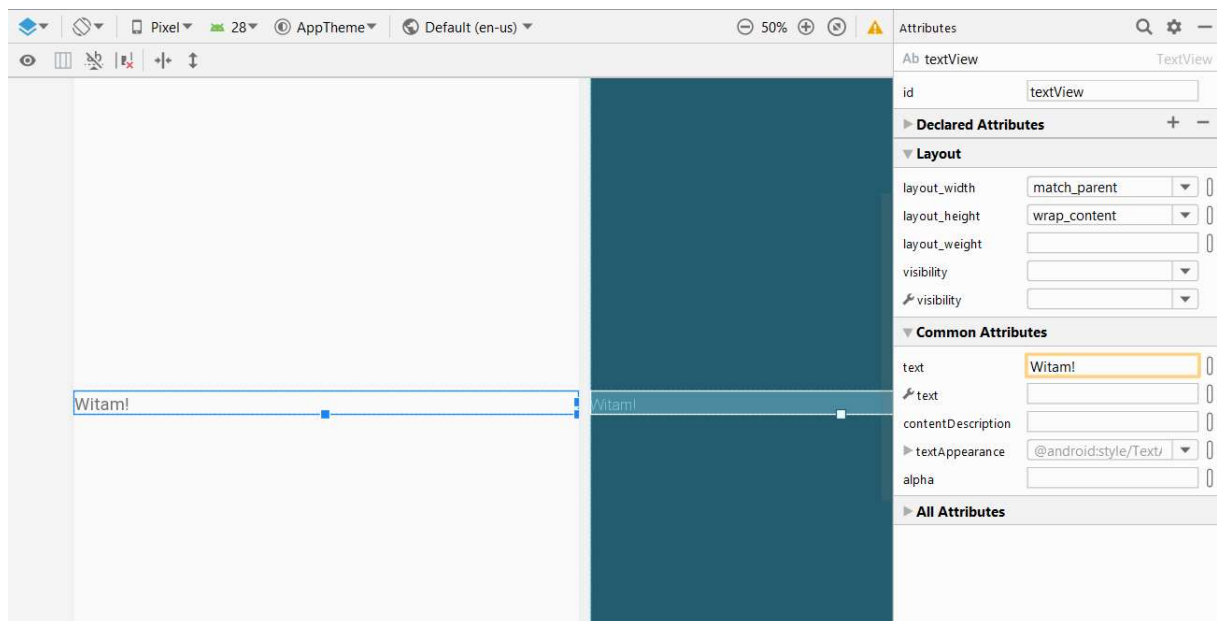
Dodawanie kontrolki do drzewa komponentów

ZADANIE

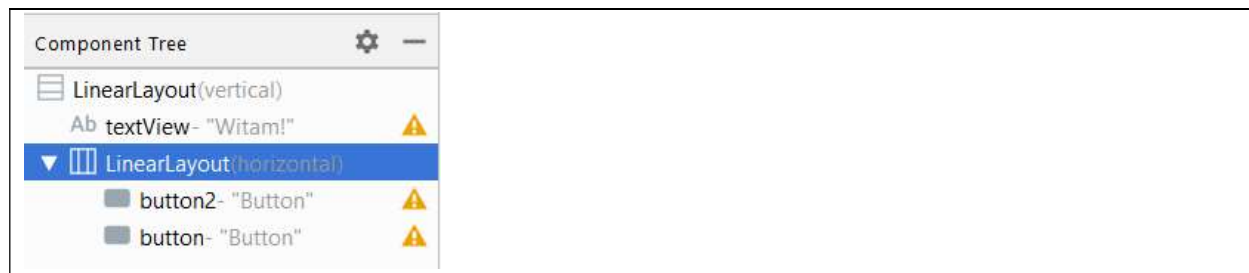
Dalej do naszego layoutu dodaj pole tekstowe **TextView**.



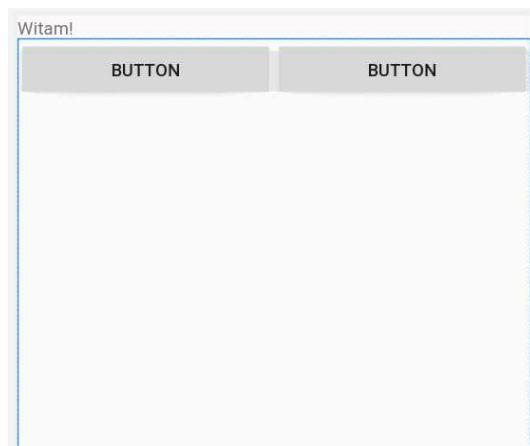
Ustaw tekst dla naszego komponentu.



Następnie dodaj kolejne kontrolki do drzewa komponentów.



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/button2"
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/button"
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 `layout_width="wrap_content"` oznacza „dopasuj do zawartości”.


ZADANIE

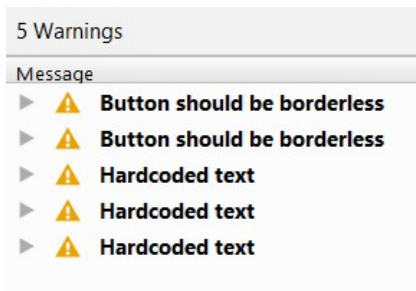
Zmodyfikuj kod layoutu do następującej postaci:

```
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="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Witam!"
13         android:padding="24dp" />
14
15     <LinearLayout
16         android:layout_width="wrap_content"
17         android:layout_height="wrap_content"
18         android:orientation="horizontal">
19
20         <Button
21             android:id="@+id/button2"
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/button"
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).

Ostrzeżenia

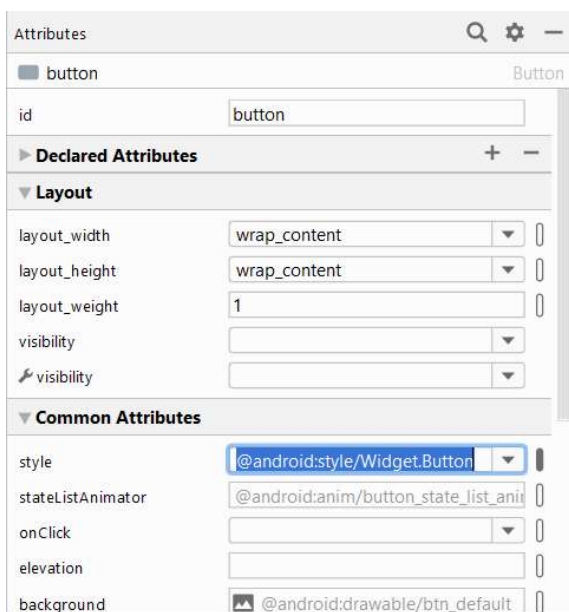
W naszym projekcie pojawiły się pewne ostrzeżenia. Aby je zobaczyć klikamy na ikonę  w górnej części Designera.



Pierwsze 2 ostrzeżenia dotyczą stylów przycisków.

ZADANIE

Zmień style przycisków w następujący sposób:



Pozostałe 3 ostrzeżenia dotyczą tekstów ustawionych na naszych komponentach. Zostały one niejako „wryte” 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 kontroltek, 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">MyApplication</string>
3     <string name="lewy">Cześć!</string>
4     <string name="prawy">Dzień dobry!</string>
5     <string name="przywitanie">Witam!</string>
6 </resources>
7
```

Następnie zmień atrybut **text** dla **TextView** w następujący sposób:

▼ Common Attributes

text

@string/przywitanie

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. button_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 android.myapplication
2
3 import ...
4
5
6
7 class MainActivity : AppCompatActivity() {
8
9     override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11         setContentView(R.layout.activity_main)
12
13         Button_czesc.setOnClickListener { it: View!
14             TextView_witam.setText("Cześć!")
15         }
16     }
17 }
18
```

W liniijkach 13-15 dodaliśmy obsługę kliknięcia na przycisk o id=Button_czesc. W wyniku tego kliknięcia zmieni się tekst wyświetlany na komponencie TextView.

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 w następujący sposób:

```
1 package android.myapplication
2
3 import ...
4
5
6
7 class MainActivity : AppCompatActivity() {
8
9     override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11         setContentView(R.layout.activity_main)
12
13
14         Button_czesc.setOnClickListener { it: View!
15             var message = Toast.makeText(applicationContext, text: "Cześć!", Toast.LENGTH_LONG)
16             message.show()
17         }
18
19         Button_dzienDobry.setOnClickListener { it: View!
20             var message = Toast.makeText(applicationContext, text: "Dzień dobry!", Toast.LENGTH_LONG)
21             message.show()
22         }
23     }
24 }
```

Przetestuj działanie aplikacji.

W linii 15 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 16 wywołujemy metodę **show()** na naszej zmiennej **message**.

ZADANIE

Dokonaj modyfikacji kodu klasy głównej wg poniższego rysunku.

```
1 package android.myapplication
2
3 import ...
4
5
6
7
8 class MainActivity : AppCompatActivity() {
9
10     override fun onCreate(savedInstanceState: Bundle?) {
11         super.onCreate(savedInstanceState)
12         setContentView(R.layout.activity_main)
13
14         Button_czesc.setOnClickListener { it: View!
15             var message = Toast.makeText(applicationContext, text: "Cześć!", Toast.LENGTH_LONG)
16             message.show() ;
17         }
18
19         Button_dzienDobry.setOnClickListener { it: View!
20             var message = Toast.makeText(applicationContext, Button_dzienDobry.text, Toast.LENGTH_LONG)
21             message.show() ;
22         }
23     }
24 }
```

Tym razem w linii 20 utworzony został Toast, na którym widniał będzie tekst pobrany z przycisku **Button_dzienDobry**.

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.