



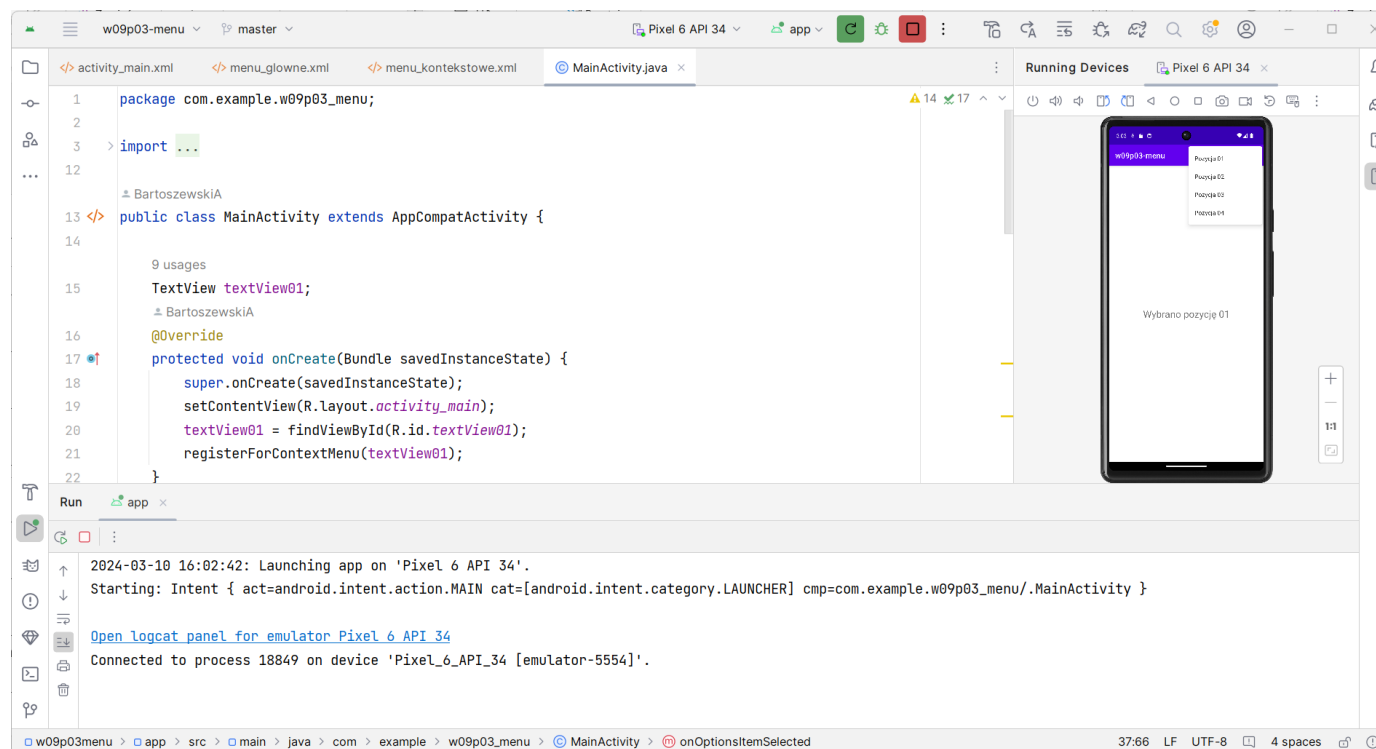
# PROGRAMOWANIE APLIKACJI MOBILNYCH

**Wykład**

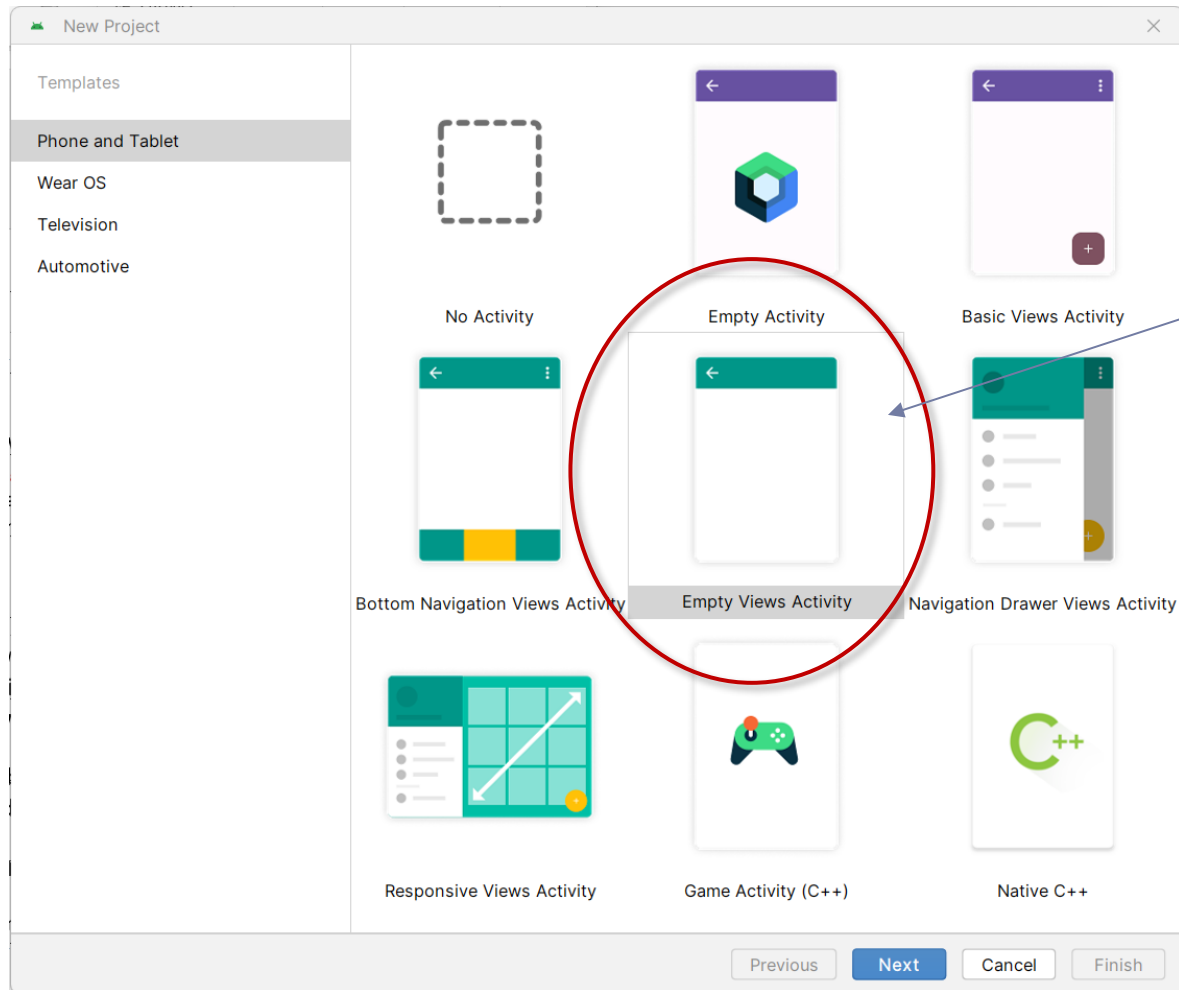
*dr Artur Bartoszewski*

## Środowisko programistyczne

<https://developer.android.com/studio/>



# Tworzenie projektu



Szablonem od którego należy rozpocząć naukę tworzenia aplikacji jest pusty szablon zawierający pojedynczą, prostą aktywność.

Wykorzystanie bardziej złożonych szablonów wymaga dużo większej wiedzy o budowie aplikacji i sposobie współdziałania jej komponentów.



# Tworzenie projektu



New Project

Empty Views Activity

Creates a new empty activity

Name: My Application

Package name: com.example.myapplication

Save location: .\023-24\2023-24\_Programowanie\_aplikacji\_mobilnych\_NST\_Wyklad\MyApplication

Language: Java

Minimum SDK: API 24 ("Nougat"; Android 7.0)

Build configuration language: Kotlin DSL (build.gradle.kts) [Recommended]

Previous Next Cancel Finish

Nazwa projektu i folder docelowy.

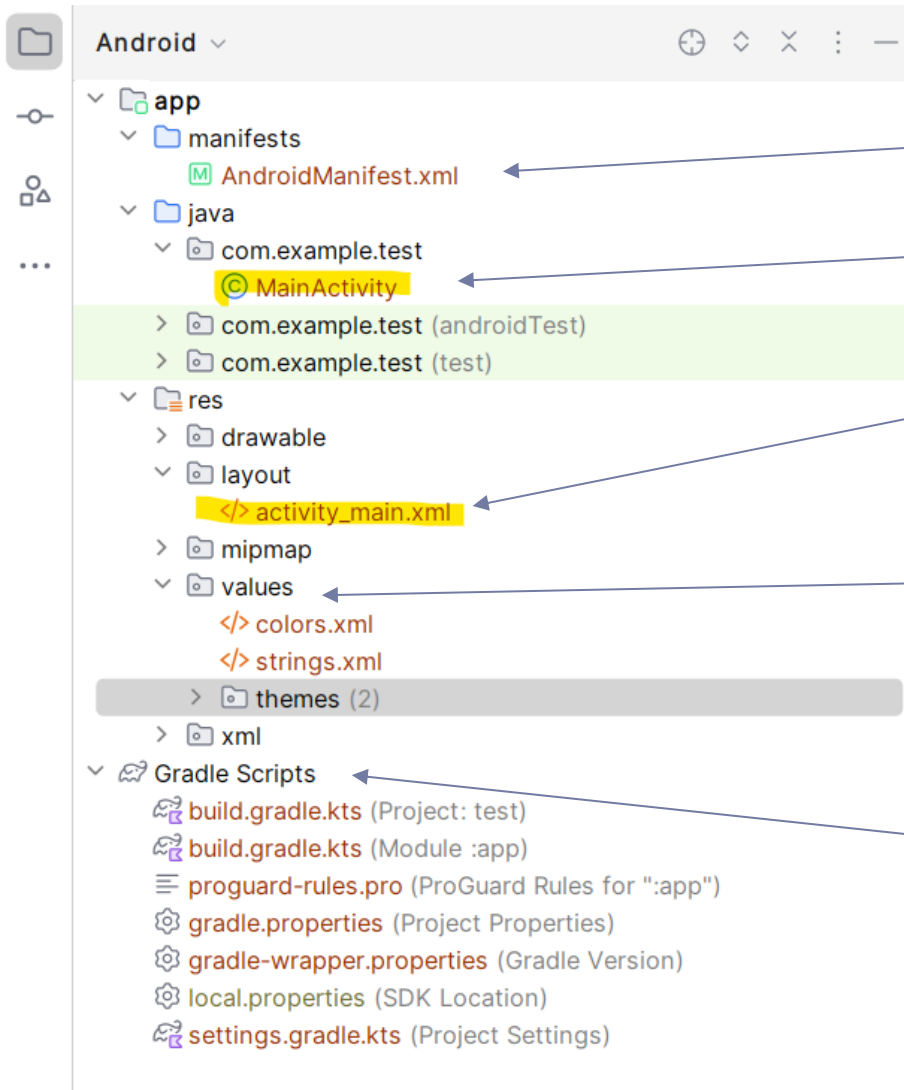
**Uwaga:** należy sprawdzić czy w ścieżce utworzony został folder dla aplikacji. Ryzykujemy że aplikacja rozsmaruje się w folderze nadrzędnym

Do wyboru są 2 języki programowania Java oraz Kotlin. **W tym wykładzie będziemy używać Javy**

Wybieramy minimalną wersję SDK czyli wersję Androida poniżej której aplikacje nie uruchomi się.

Podkreślam nie jest to wersja pod którą kompilujemy lecz tylko wpis w ustawieniach który informuje, że nie wolno uruchamiać aplikacji poniżej tej wersji.

# Struktura projektu



Plik zawierający podstawowe ustawienia aplikacji

Plik zawierający kod języka java.

Plik zawierający opis leja auto aplikacji w postaci XML

Folder zawierający zasoby czyli dodatkowe pliki dołączone do aplikacji

Skrypty konfiguruje projekt j odpowiadające za kompilację aplikacji

**AndroidManifest.xml** - Manifest – Plik sterujący, który zawiera informacje o charakterze aplikacji oraz każdym jej komponencie. Manifest wykorzystywany jest do zapisywania informacji na temat uprawnień, poszczególnych aktywności, czy też informacji na temat wykorzystywanej wersji API.

**Java** - w tym katalogu przechowywane są wszystkie nasze pliki zawierające kod Javy.

**Res** - katalog res (resources) zawiera wszystkie statyczne zasoby – obrazki, pliki dźwiękowe, video itp

**Layout** - Katalog przeznaczony na pliki xml odpowiadające za interfejs użytkownika w aplikacji (layout).

**Menu** - katalog dla plików xml definiujących menu w aplikacji.

**Minimap** - katalog przeznaczony do przechowywania ikony aplikacji.

**Values** – katalog zawierające wartości różnego rodzaju zmiennych i stałych aplikacji. Zawiera:

- colors.xml - plik w którym zdefiniowane są kolory, które później możemy użyć w różnych miejscach aplikacji.
- dims.xml - plik w którym definiujemy marginesy wykorzystywane w layoucie.
- strings.xml - plik w którym definiujemy wszystkie rzeczy tekstowe jakie będą zawarte w interfejsie użytkownika. Dzięki temu w przyszłości łatwo będziemy mogli dorobić obsługę innych języków.

Skrypty GRADLE - skrypty „budujące” aplikację.

**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”

**settings.gradle** – plik zawiera informacje o wszystkich podprojektach jakie muszą zostać skompilowane przy kompilacji aplikacji.



---

# Podstawowe kontrolki

**TextView** - służy do wyświetlania na ekranie tekstu.

```
<TextView
    android:id="@+id/textView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="To jest kontrolka \nwyświetlająca tekst na ekranie"
    android:textSize="25sp"
    android:gravity="center"
    android:textColor="#3680A1"
    android:background="#E1E9EA"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    android:padding="10dp"/>
```



<TextView

```
android:id="@+id/textView01"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:text="To jest kontrolka \nwyswietlajaca tekst na ekranie"  
android:textSize="25sp"  
android:gravity="center"  
android:textColor="#3680A1"  
android:background="#E1E9EA"  
app:layout_constraintBottom_toBottomOf="parent"  
app:layout_constraintEnd_toEndOf="parent"  
app:layout_constraintStart_toStartOf="parent"  
app:layout_constraintTop_toTopOf="parent"  
android:padding="10dp"/>
```

Nadanie widokowi Id jest konieczne, jeżeli będziemy się do niego odwoływać w kodzie Java

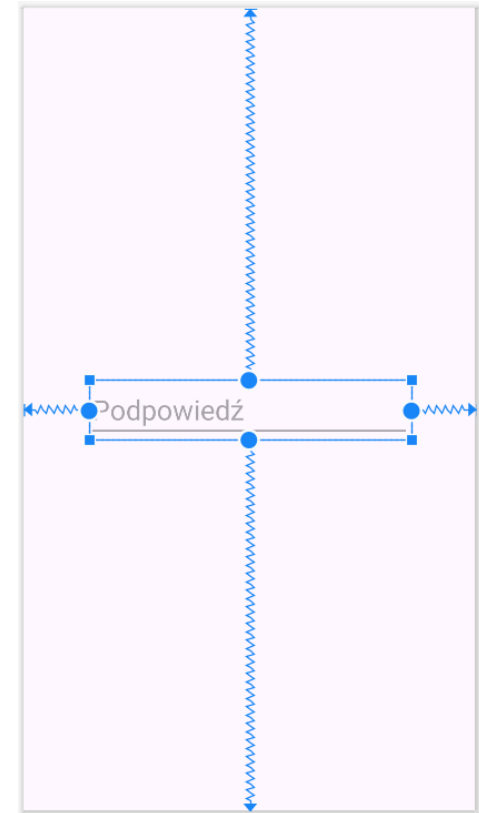
Szerokość wysokość kontrolki może być (jak w tym wypadku) dostosowana do zawartości, może wypełniać obiekt rodzica, lub też być wyliczana z innych ustawień „Odp”

Tekst może być wypisany na poziome layoutu, lecz można go modyfikować w kodzie Javy

Pozycjonowanie kontrolki zależy od użytego layoutu. W tym wypadku kontrola jaka jest wyśrodkowane na layoutcie Constraint Layout

**TextView** – pole edycyjne do wpisywania tekstu

```
<EditText
    android:id="@+id/editText01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="Podpowiedź"
    android:ems="10"
    android:text=""
    android:textSize="25sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
```



```
<EditText
    android:id="@+id/editText01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="Podpowiedź"
    android:ems="10"
    android:text=""
    android:textSize="25sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
```

Nadanie widokowi Id jest konieczne aby odczytać wpisaną wartość z poziomu Javy

Szerokość wysokość kontrolki może być (jak w tym wypadku) dostosowana do zawartości, może wypełniać obiekt rodzica, lub też być wyliczana z innych ustawień „Odp”

**Uwaga:** w niektórych wypadkach kontrolka może się rozszerzać w miarę wpisywania tekstu

Ems definiuje długo pola - jednostka oznacza liczbę liter m które mają się zmieścić w polu (ze spacjami)

W pole tekstowe zwykle tekst wpisuje Użytkownik możemy jednak wypełnić je także z poziomu xml.

```
<EditText
    android:id="@+id/editText01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="Podpowiedź"
    android:ems="10"
    android:text=""
    android:textSize="25sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
```

Nadanie widokowi Id jest konieczne aby odczytać wpisaną wartość z poziomu Javy

Szerokość wysokość kontrolki może być (jak w tym wypadku) dostosowana do zawartości, może wypełniać obiekt rodzica, lub też być wyliczana z innych ustawień „Odp”

**Uwaga:** w niektórych wypadkach kontrolka może się rozszerzać w miarę wpisywania tekstu

Ems definiuje długo pola - jednostka oznacza liczbę liter m które mają się zmieścić w polu (ze spacjami)

W pole tekstowe zwykle tekst wpisuje Użytkownik możemy jednak wypełnić je także z poziomu xml.

Kontrolki EditText i TextView przechowują wartości w postaci łańcuchów znaków

## Dostęp do kontrolki z poziomu kodu Java - ZAPIS.

Pierwszym krokiem zawsze jest utworzenie w kodzie zmiennej - referencji do obiektu typu TextView i powiązanie jej z Id widoku na layoucie

```
EditText poleTekstowe;  
poleTekstowe = findViewById(R.id.editText01);
```

Tekst wyświetlany w kontrolce zmienić możemy za pomocą metody *.setText()*

```
poleTekstowe.setText("Tekst do wstawienia");
```

Wyświetlenie wartości liczbowej wymaga prze konwertowania jej na łańcuch znaków (String).

```
int x = 100;  
poleTekstowe.setText(String.valueOf(x));
```

## Dostęp do kontrolki z poziomu kodu Java - ODCZYT.

Pierwszym krokiem zawsze jest utworzenie w kodzie zmiennej - referencji do obiektu typu TextView i powiązanie jej z Id widoku na layoucie

```
EditText poleTekstowe;  
poleTekstowe = findViewById(R.id.editText01);
```

Tekst wczytać można za pomocą metody *.getText().toString()*

```
String tekst = poleTekstowe.getText().toString();
```

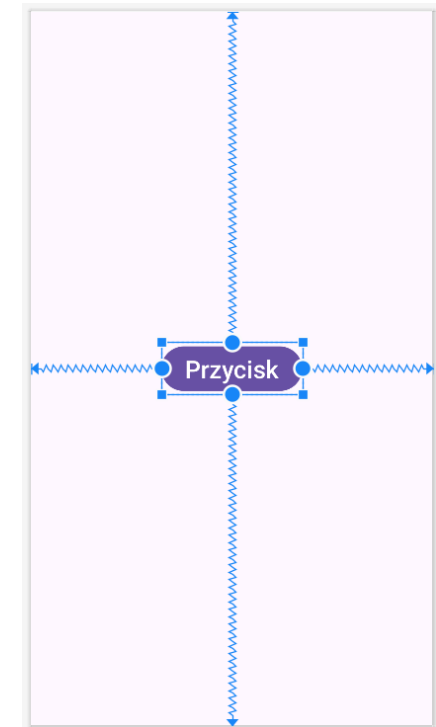
Aby wczytać liczbę należy wczytać tekst i zamienić go na liczbę

```
String tekst = poleTekstowe.getText().toString();  
Double liczba = Double.valueOf(tekst);
```

```
String tekst = poleTekstowe.getText().toString();  
int x = Integer.valueOf(tekst);
```



```
<Button
    android:id="@+id/button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Przycisk"
    android:textSize="25sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
/>
```



```
<Button
    android:id="@+id/button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Przycisk"
    android:textSize="25sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
/>
```

W przypadku omawianej na tym wykładzie metody obsługi przycisku Id nie będzie konieczne.

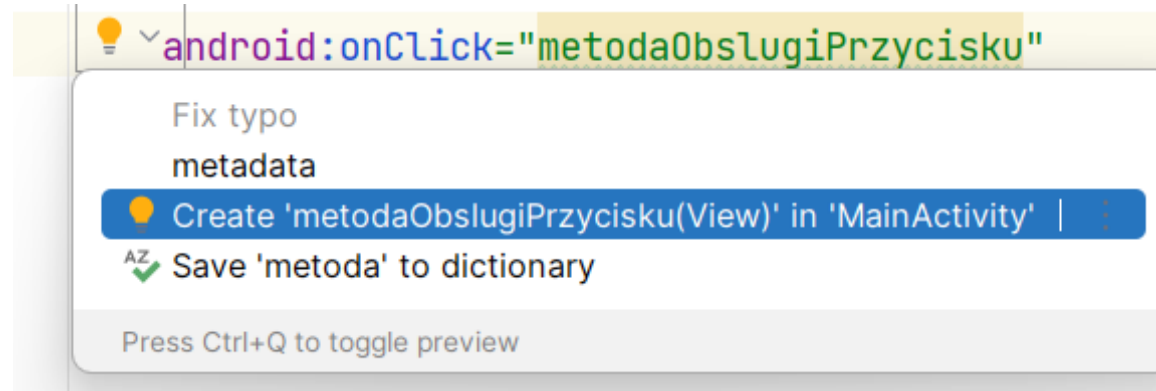
Można definiować tekst pojawiający się na przycisku i jego rozmiar. Nie powinno się używać atrybutów definiujących kolor tekstu i kolor tła przycisku.

## Obsługa przycisku za pomocą atrybutu onClick().

Kontrolce nie tylko przyciskowi przypisać można atrybut onClick() definiujący to jaka metoda ma zostać wywołana po kliknięciu na komponent.

```
android:onClick="metodaObslugiPrzycisku"
```

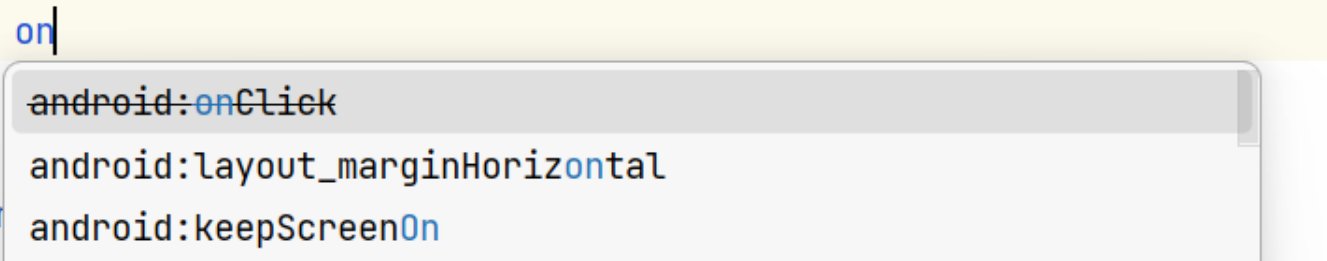
W parametrze atrybutu wpisujemy nazwę jaką chcemy nadać metodzie obsługi przycisku i w proponowanych przez środowisko akcji wybieramy utworzenie metody w Main Activity



W pliku z kodem Java utworzona zostanie metoda obsługi zdarzenia kliknięcia.

```
public void metodaObslugiPrzycisku(View view) {  
    //tu wpisz kod, który ma zostać wykonany po kliknięciu  
}
```

W tym miejscu dodać należy, że atrybut `onClick` uznany jest przez Google za przestarzały (deprecated).



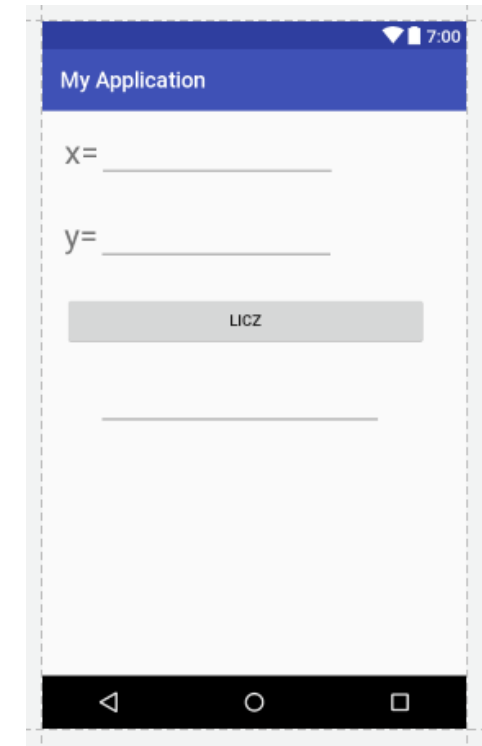
```
on|  
android:onClick  
android:layout_marginHorizontal  
android:keepScreenOn
```

Na następnym wykładzie poznamy zalecaną aktualnie metodę obsługi przycisków z wykorzystaniem słuchacza zdarzeń.

# ZADANIE PRAKTYCZNE:

Zadanie:

- aplikacja posiada dwa pola edycji w które wprowadzamy liczby
- po kliknięciu na przycisk licz są one sumowane i wypisywane w trzecim polu edycji



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:padding="20dp"
  tools:context="pl.uniwersytetradom.bartoszewski.artur.myapplication.MainActivity"

  <android.support.constraint.Guideline
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/guideline"
    app:layout_constraintGuide_begin="32dp"
    android:orientation="vertical" />

  <TextView...>

  <EditText...>

  <TextView...>

  <EditText...>

  <Button...>

  <EditText...>

</android.support.constraint.ConstraintLayout>
```

```
<TextView
```

```
    android:id="@+id/textView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="x"  
    android:textSize="30dp" />
```

```
<EditText
```

```
    android:id="@+id/editText"  
    android:layout_width="215dp"  
    android:layout_height="wrap_content"  
    android:ems="10"  
    android:inputType="number"  
    app:layout_constraintStart_toEndOf="@id/textView" />
```

```
<Button
```

```
    android:id="@+id/b_licz"  
    style="@style/Widget.AppCompat.Button"  
    android:layout_width="328dp"  
    android:layout_height="wrap_content"  
    android:text="LICZ"  
    android:layout_marginTop="28dp"  
    app:layout_constraintTop_toBottomOf="@+id/editText2"  
    android:onClick="liczenie" />
```

```
package pl.uniwersytetradom.bartoszewski.artur.myapplication;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

import static java.lang.StrictMath.sqrt;
import static pl.uniwersytetradom.bartoszewski.artur.myapplication.R.id.editText2;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

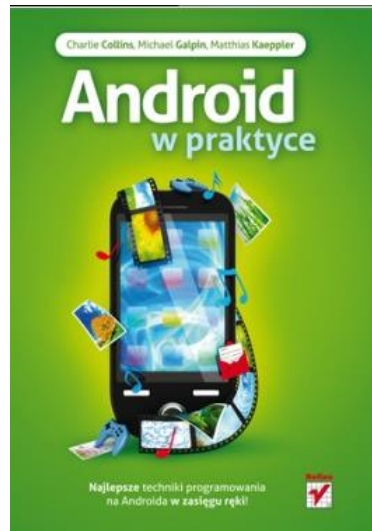
    public void liczenie (View view) {
        double x=0, y=0, wynik;
        int z;
        String a,b;
        EditText poleX = (EditText) findViewById(R.id.editText);
        EditText poleY = (EditText) findViewById(R.id.editText2);
        EditText poleWynik = (EditText) findViewById(R.id.wynik);

        a = poleX.getText().toString();
        x = Double.parseDouble(a);
        b = poleY.getText().toString();
        y = Double.parseDouble(b);

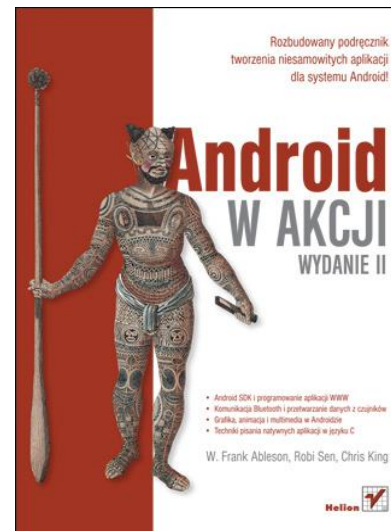
        wynik= sqrt(x*x + y*y);
        poleWynik.setText(String.valueOf(wynik));
    }
}
```



```
public void liczenie (View view) {  
    double x=0, y=0, wynik;  
    int z;  
    String a,b;  
    EditText poleX = (EditText) findViewById(R.id.editText);  
    EditText poleY = (EditText) findViewById(R.id.editText2);  
    EditText poleWynik = (EditText) findViewById(R.id.wynik);  
  
    a = poleX.getText().toString();  
    x = Double.parseDouble(a);  
    b = poleY.getText().toString();  
    y = Double.parseDouble(b);  
  
    wynik= sqrt(x*x + y*y);  
    poleWynik.setText(String.valueOf(wynik));  
}
```



<https://developer.android.com>



<https://javastart.pl/baza-wiedzy/android/>

<https://forum.android.com.pl>

