



# Programowanie Aplikacji Mobilnych na Platformę Android

## Wykład 1 – Wprowadzenie do platformy Android. Obsługiwane typy urządzeń.

Jakub Smołka



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



# Cele przedmiotu i wymagania wstępne

- **Cele przedmiotu:**
  - Poznanie zasad tworzenia aplikacji mobilnych
  - Poznanie, narzędzi służących do tworzenia aplikacji mobilnych
  - Poznanie platformy Android oraz typów obsługiwanych urządzeń
- **Wymagania wstępne:**
  - Znajomość programowania obiektowego (w języku Java lub Kotlin)
  - Język angielski – stopień podstawowy
  - Podstawowa wiedza na temat struktur danych oraz sieci komputerowych
- **Warunki zaliczenia:**
  - Egzamin pisemny w formie testu z pytaniami zamkniętymi. Próg zaliczeniowy 51%

# Efekty uczenia się

- **W zakresie wiedzy:**
  - Posiada wiedzę na temat platformy Android, typów obsługiwanych urządzeń oraz języków programowania stosowanych do tworzenia aplikacji na tę platformę
  - Zna najważniejsze elementy frameworku platformy Android
- **W zakresie umiejętności:**
  - Potrafi skonfigurować efektywne środowisko pracy programisty. Potrafi korzystać z tego środowiska
  - Potrafi zaprojektować i wykonać aplikację mobilną, dla urządzenia określonego typu, zgodnie z przyjętymi we frameworku standardami
- **W zakresie kompetencji społecznych:**
  - Jest gotów do krytycznej oceny posiadanej wiedzy w zakresie technologii mobilnych. Jest gotów do zasięgania opinii ekspertów w przypadku trudności z samodzielnym rozwiązaniem postawionych zadań

# Tematyka wykładów

|    |   |
|----|---|
| W1 | Wprowadzenie do platformy Android. Obsługiwane typy urządzeń.   |
| W2 | Środowisko programisty aplikacji urządzeń mobilnych. Dostępne narzędzia i ich możliwości.                 |
| W3 | Wstęp do tworzenia aplikacji mobilnych – elementy projektu aplikacji dla platformy Android.               |
| W4 | Tworzenie graficznego interfejsu użytkownika. Sposoby przekazywania danych pomiędzy elementami aplikacji. |
| W5 | Sposoby trwałego przechowywania danych na potrzeby aplikacji mobilnych.                                   |
| W6 | Wielozadaniowość na platformie Android. Komunikacja składników aplikacji.                                 |

# Tematyka wykładów

|     |  |
|-----|--|
| W7  | Podstawy komunikacji sieciowej na platformie Android.      |
| W8  | Wykorzystanie usług Google Play.                           |
| W9  | Modyfikacja standardowego wyglądu komponentów graficznych. |
| W10 | Wstęp do multimedialnych aplikacji na platformie Android.  |
| W11 | Wstęp do grafiki na platformie Android.                    |
| W12 | Różnice pomiędzy aplikacjami dla różnych typów urządzeń.   |

# Zalecana literatura

- **Literatura podstawowa:**
  - Strona internetowa: <https://developer.android.com>
  - Android. Wprowadzenie do programowania aplikacji. Wydanie V – Joseph Annuzzi Jr., Lauren Darcey, Shane Conder – Helion 2016
  - Programowanie aplikacji dla Androida. The Big Nerd Ranch Guide. Wydanie III – Bill Phillips, Chris Stewart, Kristin Marsicano – Helion 2017
  - Strona internetowa: <http://www.vogella.com/android.html>
  - Strona internetowa:  
[https://www.techotopia.com/index.php/Android\\_Studio\\_Development\\_Essentials\\_-\\_Java\\_Edition](https://www.techotopia.com/index.php/Android_Studio_Development_Essentials_-_Java_Edition)
  - Android i iOS – tworzenie aplikacji mobilnych, Edyta Łukasik, Maria Skublewska-Paszkowska, Jakub Smołka, Politechnika Lubelska 2014

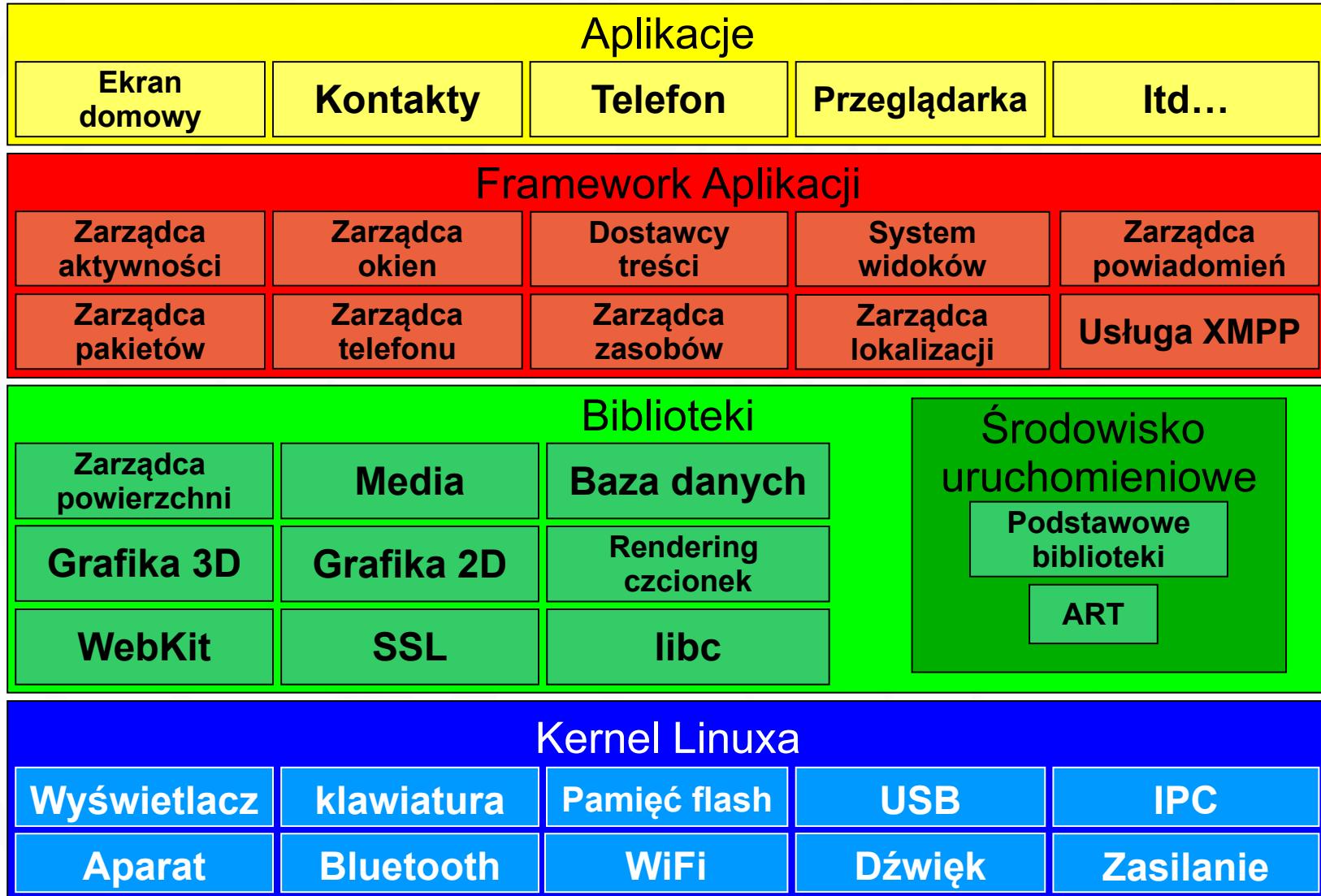
# Zalecana literatura

- Programowanie aplikacji dla systemu Android, Jakub Smołka, Politechnika Lubelska 2014
- **Literatura uzupełniająca:**
  - Strona internetowa: <https://material.io/>
  - Android na tablecie. Receptury – B.M. Harwani – Helion 2014
  - Rozpoznawanie rodzaju ruchu na podstawie odczytów sensorów urządzeń mobilnych, Jakub Smołka, Maria Skublewska-Paszkowska, Logistyka 6/2014, str. 9713–9721
  - System automatycznego wykrywania kolizji wykorzystujący urządzenia mobilne, Jakub Smołka, Edyta Łukasik, Maria Skublewska-Paszkowska, Logistyka 4/2015, str. 8315–8324

# Android – podstawowe informacje, obsługiwane urządzenia

- Android to system operacyjny rozwijany przez *Open Headset Alliance*, któremu przewodniczy *Google*
- Zbudowany został na bazie systemu *Linux* przez *Android Inc.* (Andy Rubin, Rich Miner, Nick Sears, Chris White)
- Kod źródłowy *Android Open Source Project* (AOSP) jest dostępny na licencji Apache
- Do tej pory ukazało się 12 głównych wersji systemu, jednak API dostępne dla aplikacji zmieniło się 30 razy (w sumie 31 wersji)
- Znacząca większość urządzeń z Androidem zawiera dodatkowo *Google Mobile Services* (GMS), które są dodatkiem do systemu i nie są oprogramowaniem open-source (np. Mapy Google)
- Od 2010 roku Android to najpopularniejsza platforma mobilna na świecie
- Wykorzystywany na telefonach (*Android*), tabletach (*Android*), zegarkach (*WearOS*), przystawkach TV/telewizorach (*Android TV*), samochodowych systemach informacyjno rozrywkowych (*Android Automotive*)

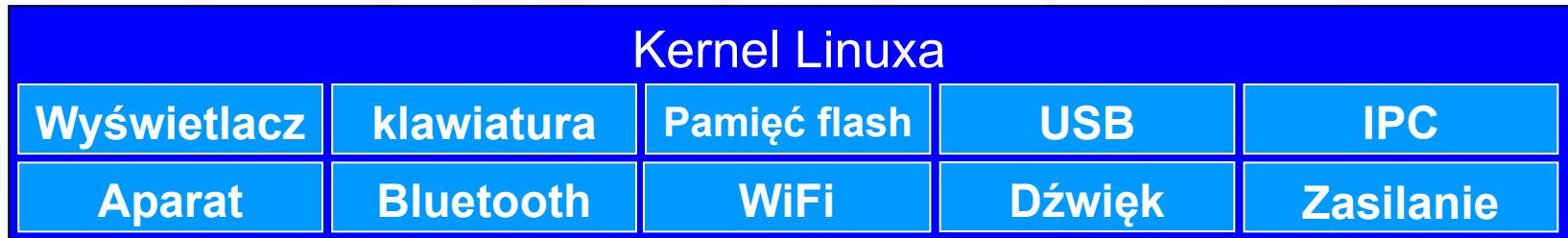
# Architektura systemu Android



źródło: <https://www.techplayon.com/android-os-architecture/>

# Android – warstwa pierwsza

- Zmodyfikowane jądro systemu *Linux*. Realizuje ono funkcje typowe dla systemu operacyjnego:
  - zarządzanie pamięcią
  - zarządzanie procesami, separację procesów, obsługa komunikacji międzyprocesowej
  - obsługę systemu plików
  - zapewnia obsługę sprzętu (wyświetlacz, klawiatura, pamięć masowa, aparat itd.)



# Android – warstwa druga

- Natywne biblioteki Androida. Zapewniają:
  - realizację funkcji graficznych
  - prostą bazę danych SQL (*SQLite*)
  - obsługę szyfrowania
  - *WebKit* - renderowanie stron WWW
- Obok bibliotek natywnych znajdują się:
  - standardowe biblioteki języka Java (podzbiór OpenJDK)
  - środowisko uruchomieniowe *ART*



# Android – warstwa trzecia

- Elementy frameworku systemu Android odpowiedzialne za:
  - zarządzanie widokami (komponentami graficznymi), oknami, zasobami
  - zarządzanie aktywnościami (podstawowymi elementami aplikacji) i pakietami
  - zarządzanie funkcjami telefonicznymi
  - zarządzanie powiadomieniami
  - dostarczaniem treści (np. kontaktów z książki adresowej, multimedialnych) aplikacjom
  - dostarczaniem usług lokalizacyjnych
  - komunikację z użyciem XMPP



# Android – warstwa czwarta

- Warstwa aplikacji. Realizują/umożliwiają:
  - obsługę ekranu domowego (launcher)
  - zarządzanie kontaktami
  - obsługę funkcji telefonicznych
  - przeglądanie internetu
  - ...
- Warto zauważyć, że wbudowane aplikacje Google/twórcy urządzenia mobilnego mogą być zastąpione przez użytkownika



# Android – środowisko uruchomieniowe

- *Android Runtime (ART)* zostało wprowadzone w Androidzie 4.4 jako *technology-preview* i stało się standardem w Android 5.0
- ART wykorzystuje komplikację z wyprzedzeniem. Cały kod kompilowany jest do kodu natywnego w momencie instalacji
- Dodatkowo usprawniono (względem maszyny *Dalvik*) alokację pamięci, działanie odśmiecacza, wprowadzono nowe możliwości debugowania i usprawniono profilowanie
- ART wykorzystuje standardowe pliki *dex* (wsteczna kompatybilność). Są konwertowane na format ELF (Executable and Linkable format)
- Wady ART to: dłuższy czas instalacji, skompilowane pliki zajmują więcej pamięci

# Android – model zabezpieczeń

- *Osobne konta* – dla każdej aplikacji w systemie tworzone jest osobne konto użytkownika. Linux zapewnia separację procesów i plików należących do różnych aplikacji. Linux zapewnia komunikację procesów
- *System uprawnień* – każda aplikacja otrzymuje minimalny zestaw uprawnień niezbędnych do działania
  - od *Androida 6.0* – dodatkowe uprawnienia przyznawane są na bieżąco. Aplikacja prosi o uprawnienia gdy ich potrzebuje
- *Podpisywanie aplikacji* – każda instalowana na urządzeniu aplikacja musi być podpisana. Pozwala to na identyfikowanie autora aplikacji



# Programowanie Aplikacji Mobilnych na Platformę Android

**Wykład 2 – Środowisko programisty aplikacji  
urządzeń mobilnych. Dostępne narzędzia  
i ich możliwości.**

Jakub Smołka



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



# Android Studio – podstawowe informacje

- Środowisko zbudowane na bazie *IntelliJ IDEA*
- Dostępne na <https://developer.android.com/studio>
- Obsługuje języki *Java*, *Kotlin*, *C++* a dzięki wtyczkom np. *Dart*
- Umożliwia tworzenie aplikacji w *Android SDK* i *NDK* oraz (dzięki wtyczce) we frameworku *Flutter*
- Cechy: wieloplatformowość, zaawansowany edytor kodu, elastyczny system budowania *Gradle*, szybki emulator urządzeń, narzędzie *Lint* wykrywające problemy związane z wydajnością, użytecznością, kompatybilnością itp., wsparcie dla platformy chmurowej *Google*

# Android Studio – wymagania

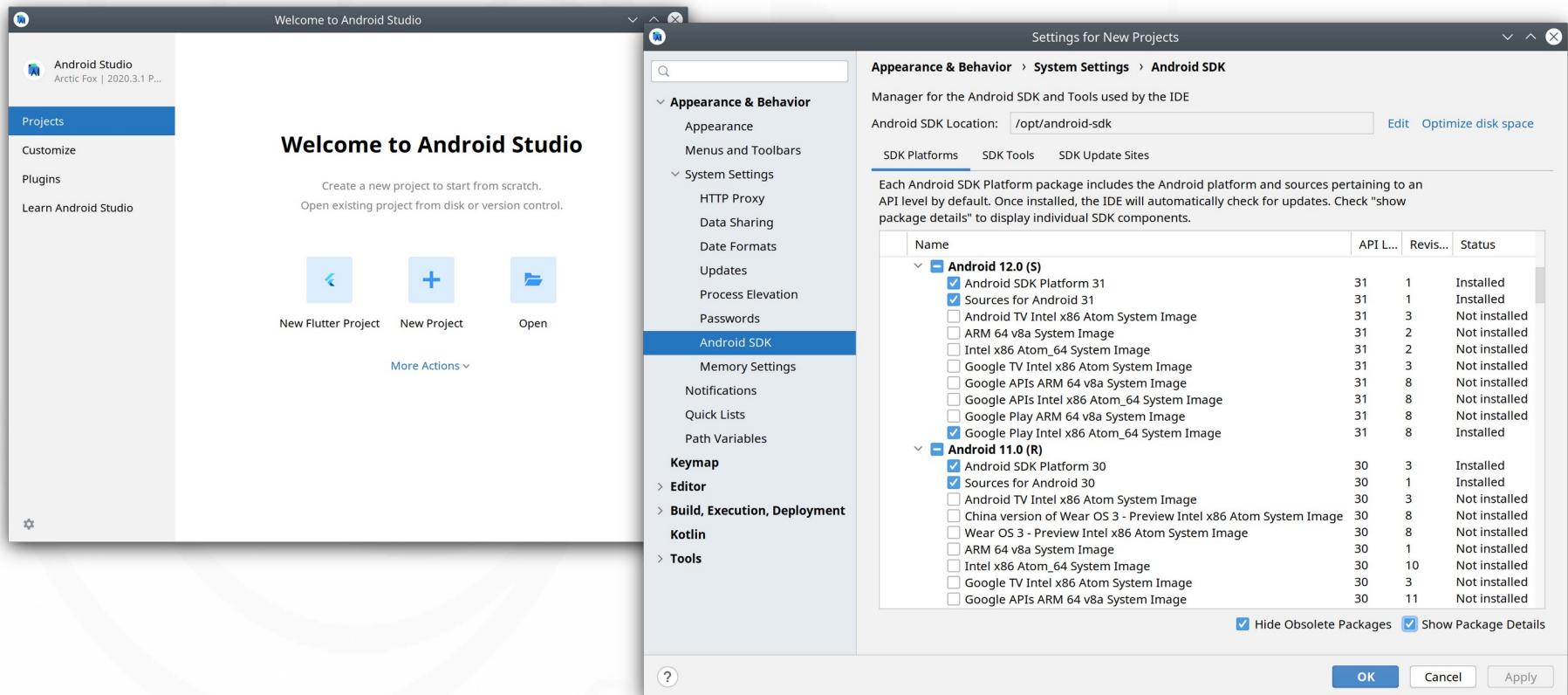
- *Ogólnie*: 8 GB RAM, 8 GB przestrzeni dyskowej (IDE + Android SDK + Android Emulator), rozdzielcość: 1280x800
- *Linux*: 64-bitowa dystrybucja z Gnome, KDE lub Unity DE, biblioteka GNU C Library (glibc) 2.31 lub nowsza, architektura x86\_64 CPU, procesor AMD z AMD-V i SSSE3 lub Intel Core (>= gen. 2)
- *Windows*: 64-bitowe Windows 8/10; procesor AMD lub Intel Core ze wsparciem hypervisor dla systemu Windows
- *MacOS*: MacOS 10.14 (Mojave) lub nowszy, procesor Intel Core (>= gen. 2) lub ARM ze wsparciem Hypervisor.Framework
- *ChromeOS*: 4GB przestrzeni dyskowej, Intel i5 (seria U) lub wyższe

# Android Studio – instalacja

- Instalacja Android Studio jest prosta – polega na pobraniu i rozkompresowaniu odpowiedniego pliku ze strony:  
<https://developer.android.com/studio>
- Linux:
  - pobrać plik android-studio-wersja-linux.zip
  - rozkompresować plik
  - przejść do katalogu `sciezka_instalacji/android-studio/bin` i uruchomić skrypt `studio.sh`
- Windows:
  - pobrać i uruchomić instalator `android-studio-wersja-windows.exe`
  - Przy pierwszym uruchomieniu uruchamiany jest kreator konfiguracji, który pobierze i m.in. zainstaluje SDK
  - Wskazane jest dodanie do zmiennej systemowej PATH katalogów: `sciezka_sdk/platform-tools` i `sciezka_sdk/tools` (typowe SDK jest instalowane w `katalog_domowy/android/sdk` (linux) lub `katalog_domowy\AppData\Local\Android\Sdk` (Windows))

# Android Studio – instalowanie różnych wersji Androida

- uruchomić Android Studio | na ekranie powitalnym wybrać *More Actions | SDK Manager*

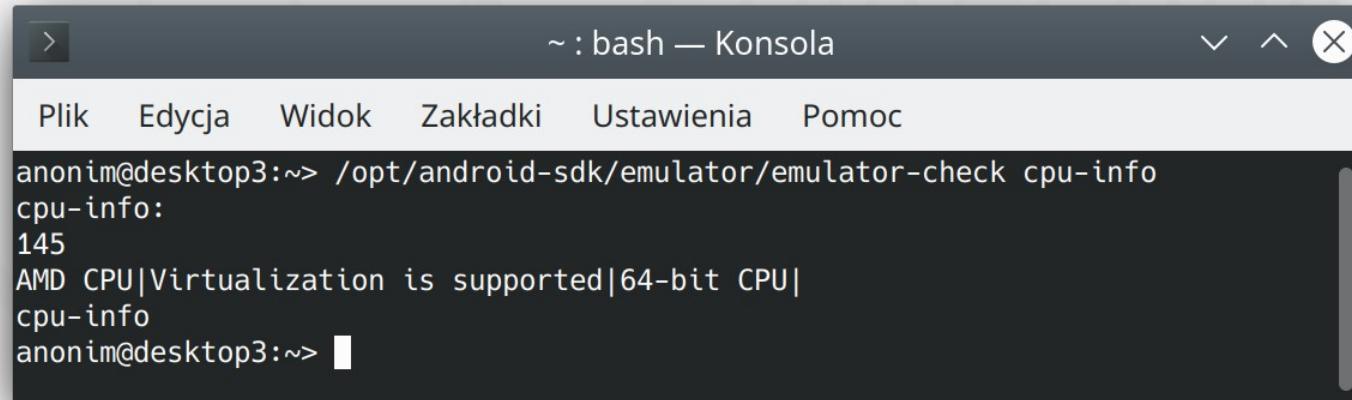


# Wymagania hypervisorów

- *KVM*: Procesor AMD z *Secure Virtual Machine* (SVM, inna nazwa: *AMD-V*) lub 64-bitowy procesor Intel z *Virtualization Technology* (*VT*) i *Execute Disable Bit* (*XD*)
- *HAXM*: 64-bitowy procesor Intel z *VT-x* i *XD*; Windows 10, Windows 8, lub Windows 7 (64 lub 32-bitowe); wyłączone Hyper-V
- *Android Emulator Hypervisor Driver for AMD Processors*: procesor AMD z technologią SVM; 64-bitowe Windows 10, Windows 8, lub Windows 7; wyłączone Hyper-V
- *Windows Hypervisor Platform (Hyper-V/WHPX)*: procesor AMD z SVM (zalecany Ryzen) lub procesor Intel z *VT-x*, *Extended Page Tables (EPT)*, *Unrestricted Guest (UG)*; Windows 10 z aktualizacją April 2018 lub nowszą

# Sprawdzanie procesora

- Sprawdzanie dostępności wirtualizacji:
  - Polecenie: emulator-check cpu-info
- Typowe położenie:
  - Linux: katalog domowy/android/sdk/emulator
  - Windows: katalog domowy\AppData\Local\Android\Sdk\emulator



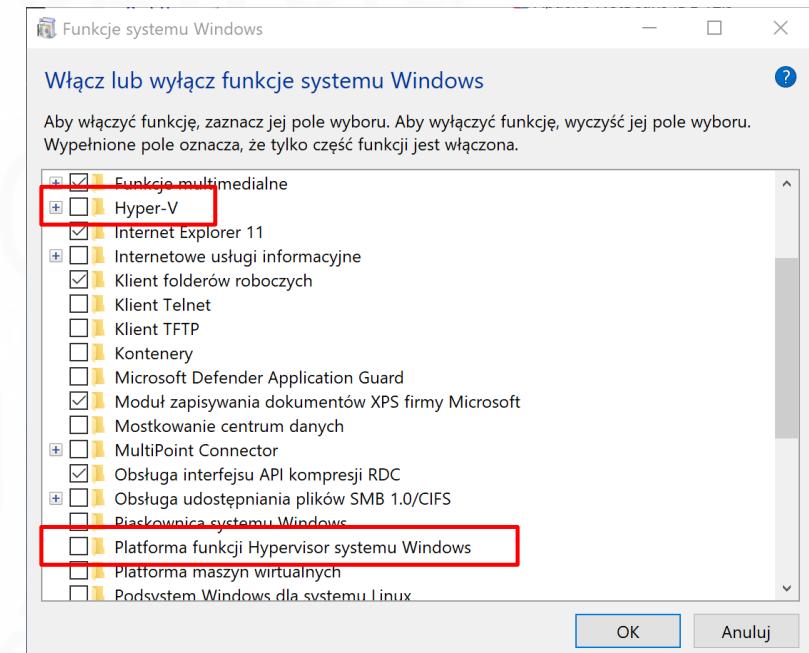
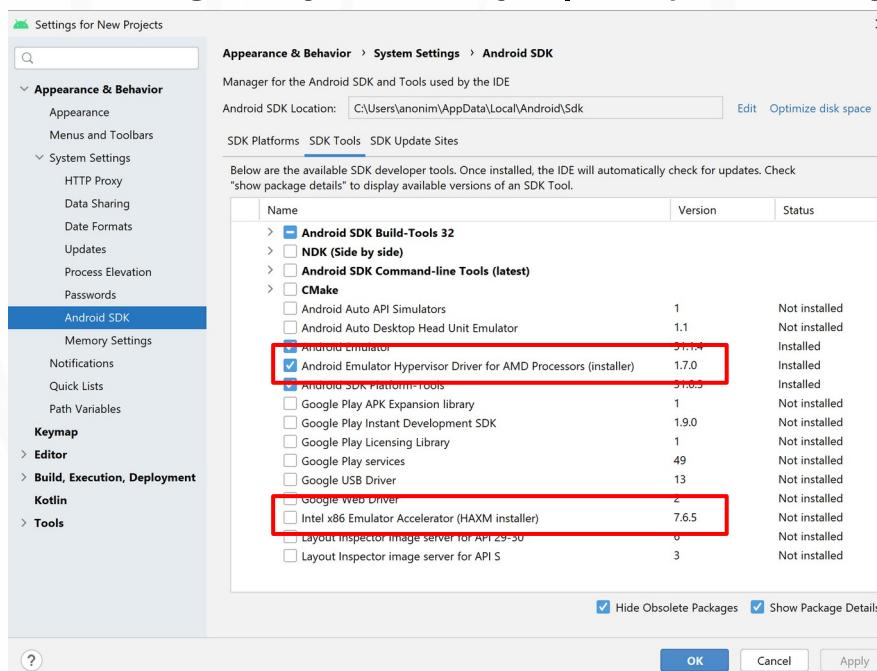
A screenshot of a terminal window titled "bash — Konsola". The window has a dark theme with light-colored text. At the top, there is a menu bar with Polish labels: "Plik", "Edycja", "Widok", "Zakładki", "Ustawienia", and "Pomoc". Below the menu, the terminal prompt shows the user's name and the path: "anonim@desktop3:~> /opt/android-sdk/emulator/emulator-check cpu-info". The output of the command is displayed, indicating that CPU virtualization is supported: "cpu-info: 145 AMD CPU|Virtualization is supported|64-bit CPU|cpu-info". The terminal ends with the user's name and the path again: "anonim@desktop3:~>".

# Instalacja Hypervisorów

- Jeżeli wirtualizacja nie została wykryta:
  - Wirtualizacja może być wyłączona w *BIOSie/UEFI* (często ustawienie domyślne)
  - *Hyper-V* może być już uruchomione
  - Procesor może nie wspierać wirtualizacji lub *SLAT* ([www.cpu-world.com](http://www.cpu-world.com))
- Linux:
  - Instalacja *KVM* zależy od dystrybucji - przykładowo w *OpenSUSE*: *Yast* | *Zarządzanie oprogramowaniem* | zainstalować wzorzec *KVM Virtualization Host*
  - Nie jest konieczna konfiguracja mostu sieciowego (jak przy typowej instalacji *KVM*)

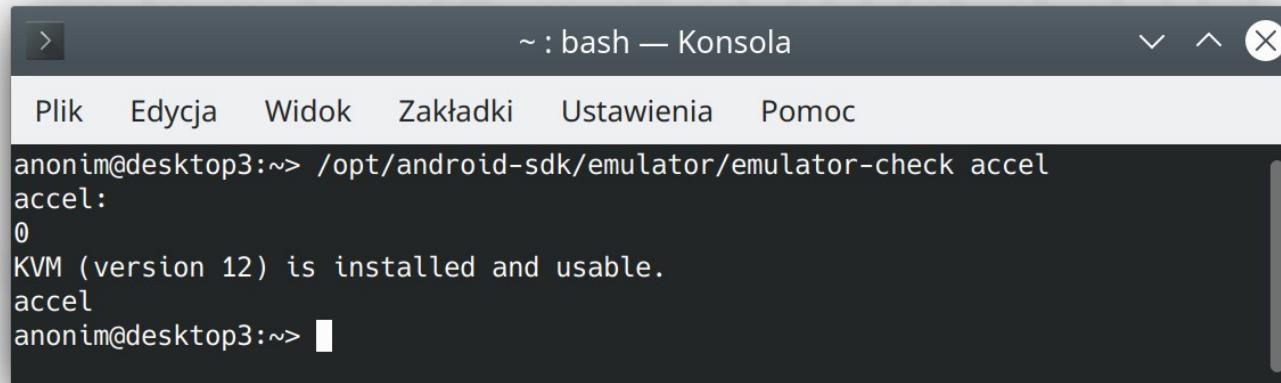
# Instalacja Hypervisorów

- Windows:
  - *Android Emulator Hypervisor Driver for AMD Processors / Intel x86 Emulator Accelerator (HAXM)*: uruchomić Android Studio | na ekranie powitalnym wybrać More Actions | SDK Manager | SDK Tools
  - *Windows Hypervisor Platform (Hyper-V)*: Panel Sterowania | Programy i funkcje | Włącz lub wyłącz funkcje systemu Windows



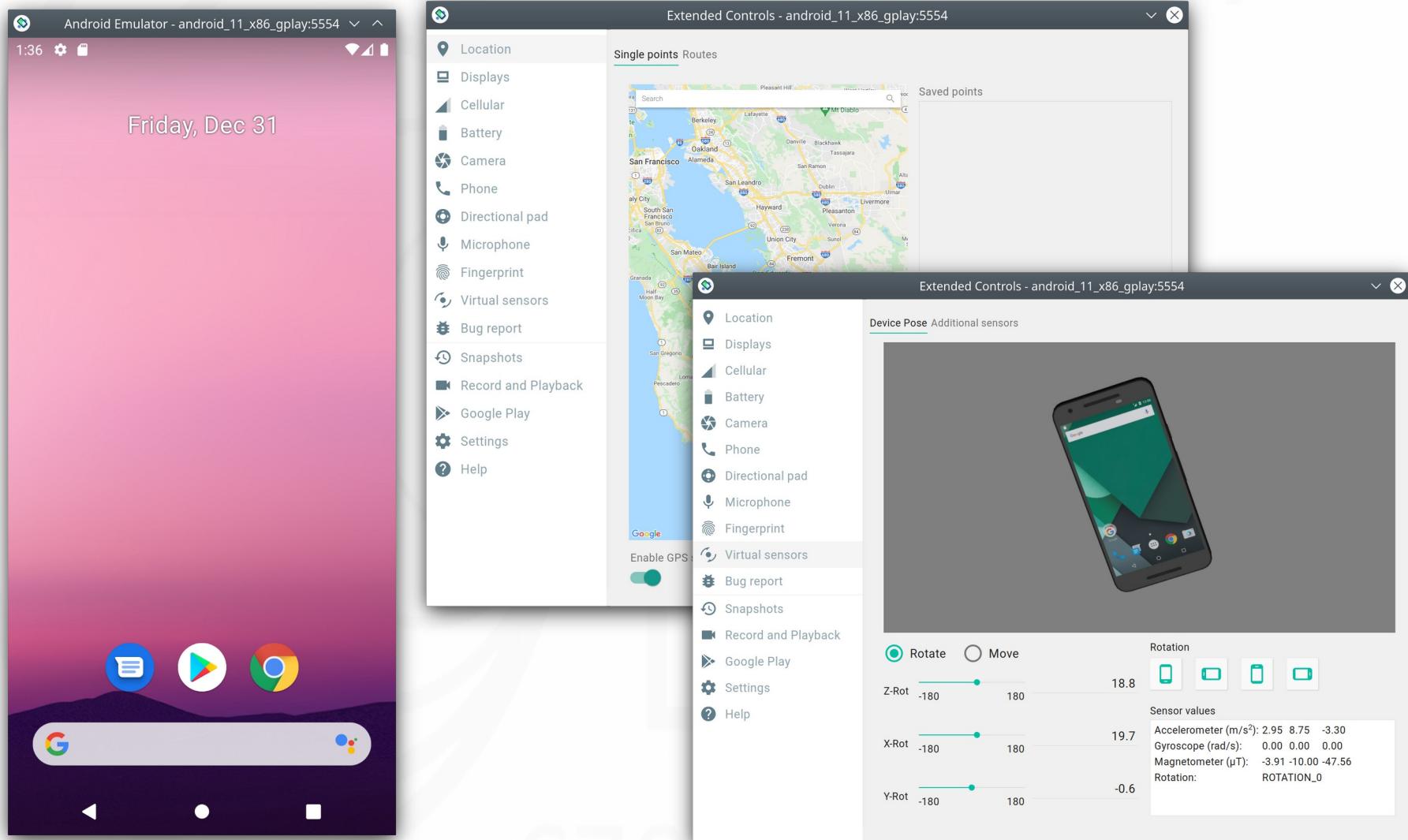
# Sprawdzanie hypervisora

- Sprawdzanie dostępności hypervisora:
- Polecenie: emulator-check accel
- Polecenie (w przypadku *Hyper-V*): emulator-check hyper-v
- Typowe położenie:
  - Linux: katalog domowy/android/sdk/emulator
  - Windows: katalog domowy\AppData\Local\Android\Sdk\emulator



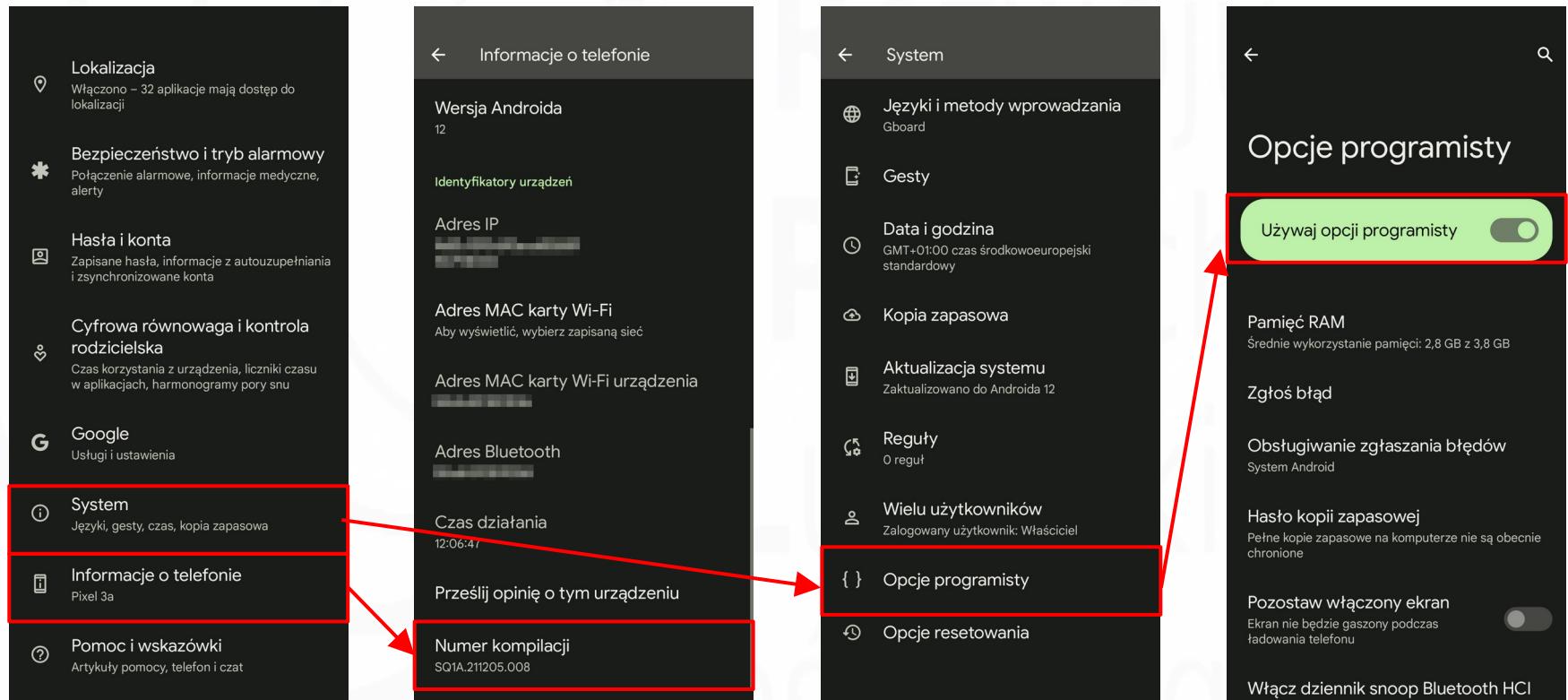
```
~ : bash — Konsola
Plik Edycja Widok Zakładki Ustawienia Pomoc
anonim@desktop3:~> /opt/android-sdk/emulator/emulator-check accel
accel:
0
KVM (version 12) is installed and usable.
accel
anonim@desktop3:~>
```

# Emulator – możliwości



# Fizyczne urządzenie – konfiguracja urządzenia

- Należy uaktywnić w urządzeniu tryb debugowania/opcje programisty.
- Dokładny sposób zależy od konkretnego urządzenia



# Fizyczne urządzenie – konfiguracja komputera

- Linux:
  - Aby ADB mogło komunikować się z podłączonym konieczne są reguły tworzenia urządzeń w systemie plików udev
  - Najprostszym rozwiązaniem jest instalacja pakietu android-udev-rules dostępnego w wielu dystrybucjach (np. [https://software.opensuse.org/package/android-udev-rules?search\\_term=android-udev-rules](https://software.opensuse.org/package/android-udev-rules?search_term=android-udev-rules)) lub pobranie go z <https://github.com/M0RF30/android-udev-rules>
- Windows:
  - Należy zainstalować sterowniki urządzenia
  - Lista odnośników do sterowników wszystkich znaczących producentów urządzeń z Androidem jest dostępna pod adresem: <https://developer.android.com/studio/run/oem-usb>

# Android Debug Bridge

- *Android Debug Bridge* jest uniwersalnym narzędziem działającym w linii poleceń. Pozwala na komunikację z fizycznym/wirtualnym urządzeniem.
- Pozwala na
  - instalowanie, debugowanie aplikacji,
  - dostęp do powłoki systemu,
  - kopiowanie plików
  - tworzenie kopii zapasowych
  - ...
- Polecenie adb znajduje się w katalogu `sdk/platform-tools/`

# Android Debug Bridge

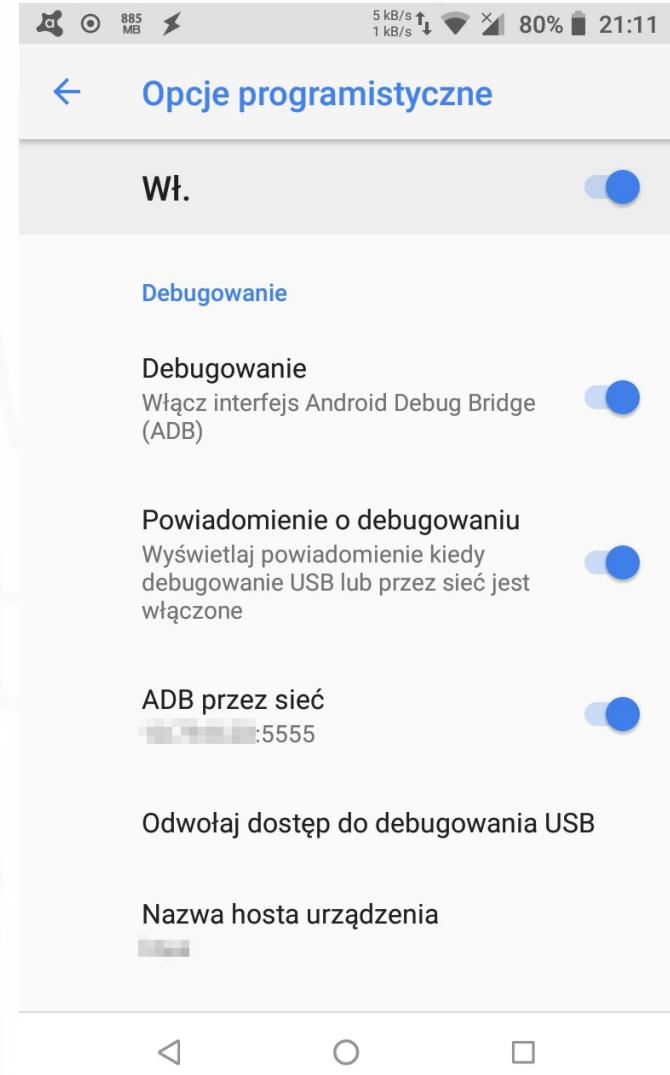
- Składa się z 3 komponentów
  - Klienta wysyłającego polecenia (polecenie adb na komputerze)
  - Demona (adb) działającego w tle na urządzeniu z Androidem. Demon wykonuje polecenia na urządzeniu
  - Serwera zarządzającego komunikacją pomiędzy klientem a demonem. Serwer pracuje w tle na komputerze
- Po uruchomieniu klient sprawdza czy proces serwera działa. Jeżeli nie to go uruchamia
- Proces serwera oczekuje na połączenia na porcie 5037/tcp. Po uruchomieniu rozpoczyna poszukiwanie emulatorów sprawdzając porty 5555 – 5585 (każdy emulator używa dwóch portów – jednego dla konsoli, drugiego dla adb np. `emulator1` używa 5554 (konsola) i 5555 (adb), `emulator2` 5556 i 5557...). Następnie oczekuje na polecenia klientów

# Android Debug Bridge

- adb devices – wyświetla listę (fizycznych/emulowanych) urządzeń
- adb shell – dostęp do powłoki systemu
- adb push plik\_lokalny katalog\_zdalny – wysłanie pliku na urządzenie
- adb pull plik\_zdalny katalog\_lokalny – skopiowanie pliku z urządzenia
- adb install aplikacja.apk – instalowanie aplikacji
- adb logcat – wyświetlanie dziennika
- adb root – restart adbd z uprawnieniami roota
- adb -s identyfikator polecenie – gdy dostępnych jest więcej urządzeń

# Android Debug Bridge

- Niektóre urządzenia umożliwiają debugowanie przez sieć
- W opcjach programistycznych należy włączyć dostęp przez ADB a następnie ADB przez sieć
- W wierszu poleceń wykonać:  
aby połączyć:  
`adb connect adres_ip_urządzenia`  
aby rozłączyć:  
`adb disconnect`  
`adres_ip_urządzenia`
- Polecenie adb znajduje się w katalogu `sdk/platform-tools/`





# Programowanie Aplikacji Mobilnych na Platformę Android

**Wykład 3 – Wstęp do tworzenia aplikacji mobilnych.  
Elementy projektu aplikacji dla platformy Android.**

**Jakub Smołka**



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój

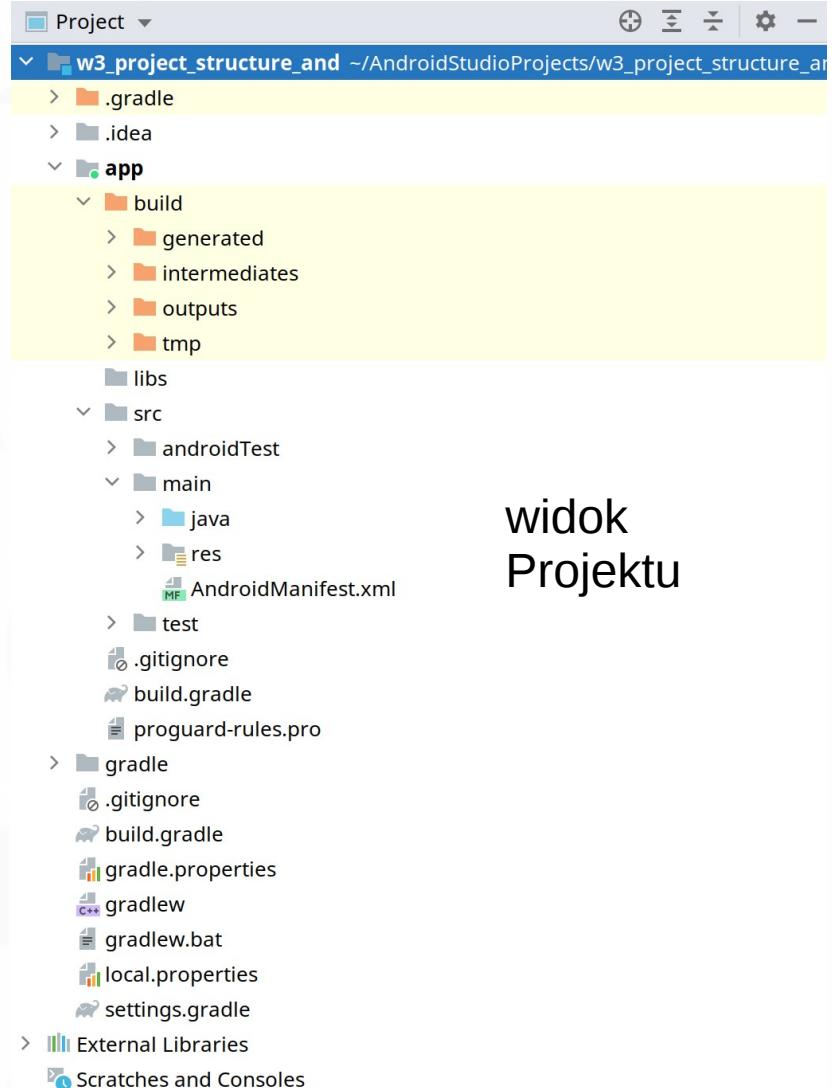
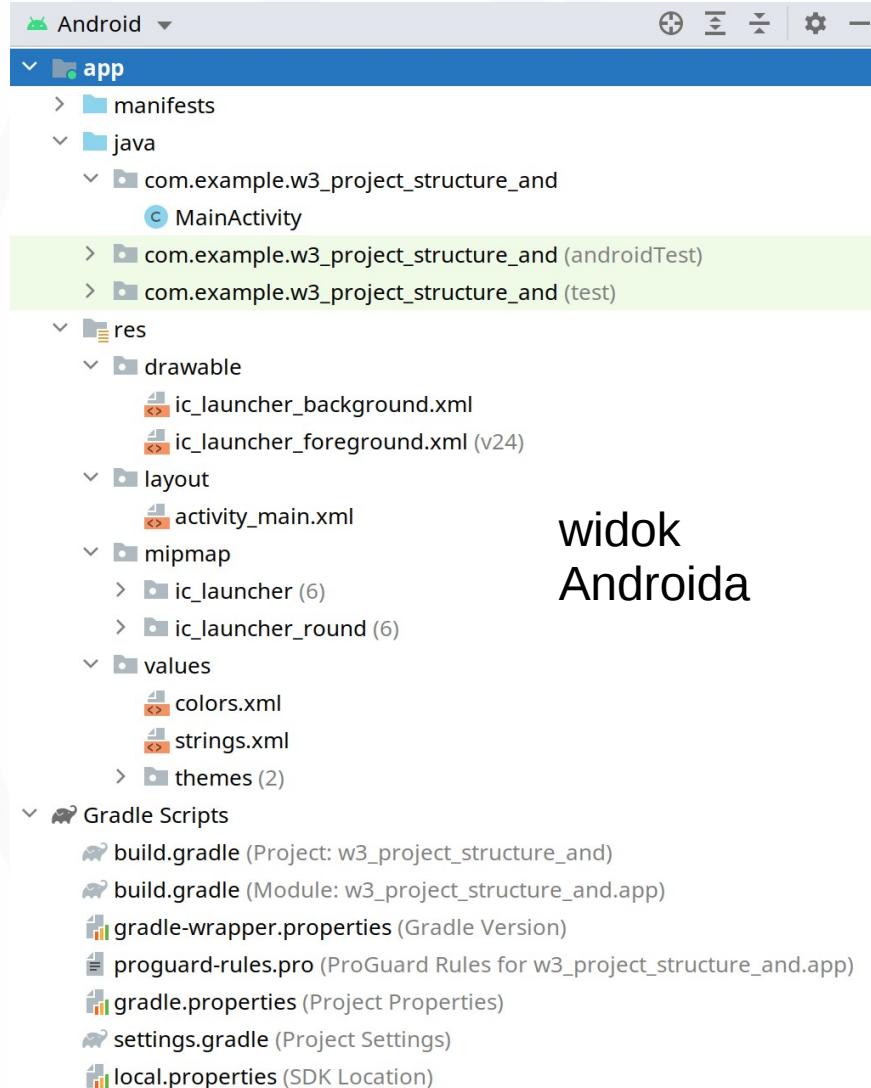


**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



# Struktura projektu – widoki projektu

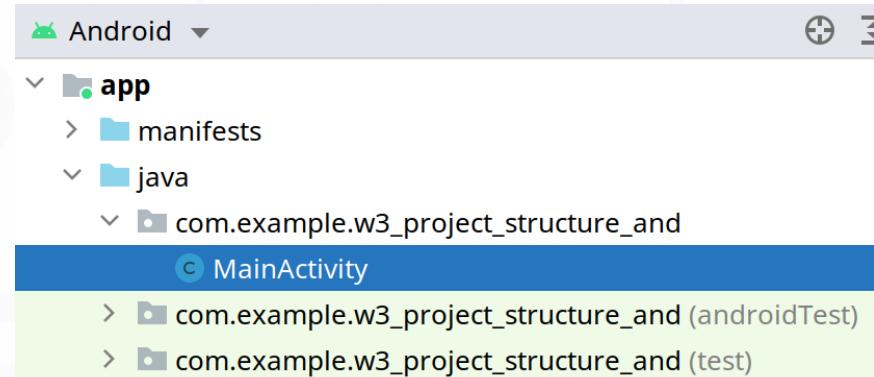


# Struktura projektu – katalogi

- app/src (java w widoku Androida) – pliki źródłowe .java
- app/build – pliki binarne (m.in. pliki apk)
- app/build/generated – pliki wygenerowane przez narzędzia z SDK
- app/src/assets – zasoby przechowywane w postaci „surowej” (trzeba samodzielnie dodać)
- app/libs – biblioteki (pliki .jar)
- app/src/res (res w widoku Androida) – zasoby (elementy graficzne, układy elementów, napisy)
- app/src/main/AndroidManifest.xml – opis aplikacji i jej składników
- build.gradle – (dwa pliki: jeden w katalogu projektu, drugi w katalogu modułu np. app) – właściwości projektu (np. docelowa platforma, zależności)

# Struktura projektu – pliki źródłowe

- Poniżej przedstawiono automatycznie wygenerowaną pustą aktywność aplikacji – plik `MainActivity.java`



```
package com.example.w3_project_structure_and;

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

public class MainActivity extends AppCompatActivity {

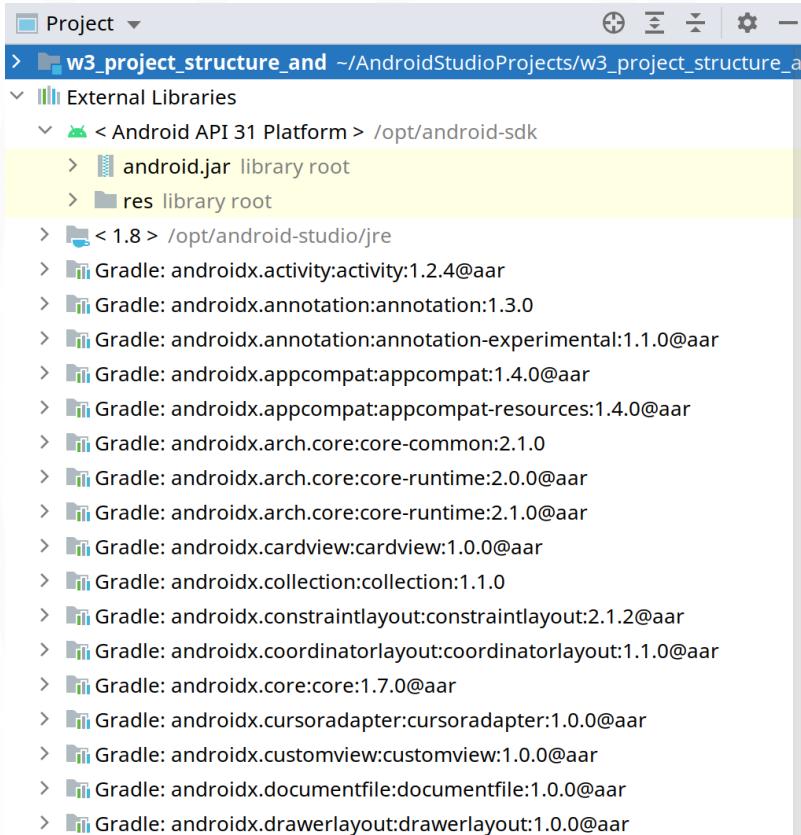
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

# Struktura projektu – klasa R (generated)

- Obecne wersja narzędzi SDK generuje klasę R od razu w postaci binarnej. Poniżej przedstawiono fragment R.java ze starszej wersji

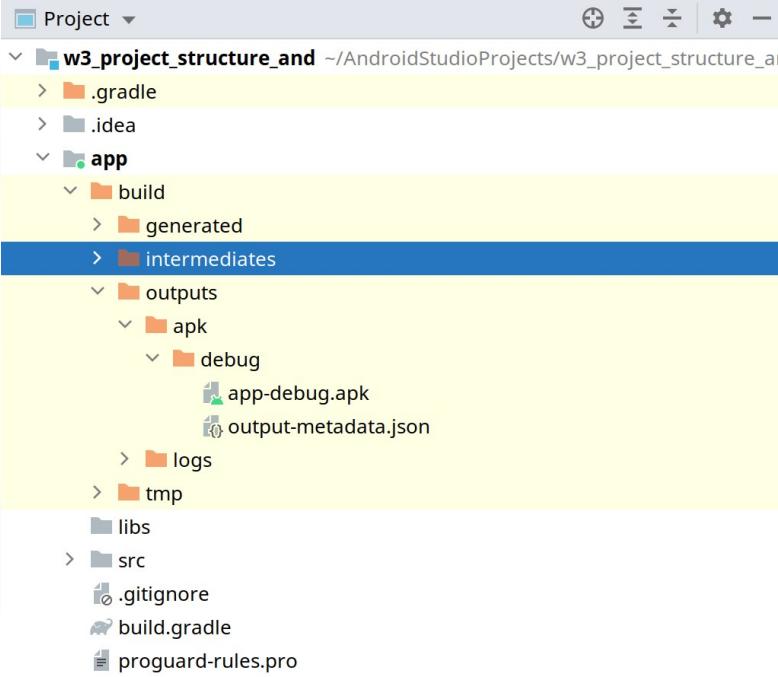
```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */
package com.example.structure.structure;
public final class R {
    public static final class anim {
        public static final int abc_fade_in=0x7f050000;
        public static final int abc_fade_out=0x7f050001;
        public static final int
            abc_grow_fade_in_from_bottom=0x7f050002;
        public static final int abc_popup_enter=0x7f050003;
        public static final int abc_popup_exit=0x7f050004;
        public static final int
            abc_shrink_fade_out_from_bottom=0x7f050005;
        public static final int abc_slide_in_bottom=0x7f050006;
        public static final int abc_slide_in_top=0x7f050007;
        public static final int abc_slide_out_bottom=0x7f050008;
        public static final int abc_slide_out_top=0x7f050009;
    }
    public static final class attr {
```

# Struktura projektu – biblioteki



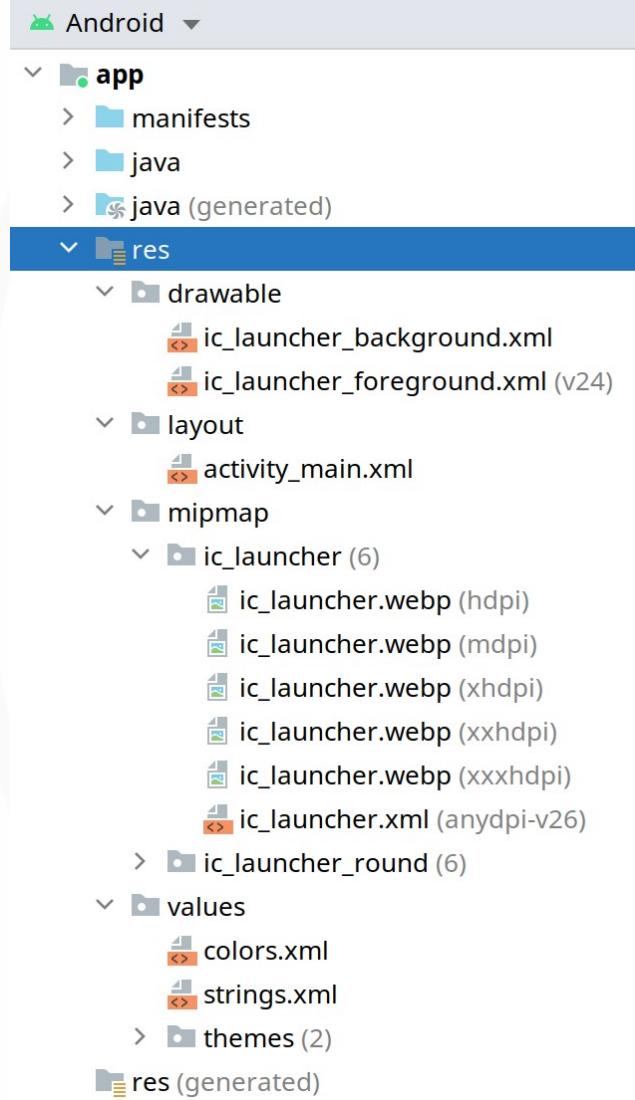
- `android.jar` – standardowa biblioteka Androida
- `1.8` – standardowa biblioteka Javy
- `androidx` – biblioteki składające się na Android Jetpack – zbiór bibliotek, które promują stosowanie zalecanych wzorców i zapewniają wsteczną kompatybilność z wcześniejszymi wersjami Androida
- `material` – komponenty Material Design,
- `junit, hamcrest` – testy

# Struktura projektu – biblioteki



- **outputs** – skompilowany projekt (pliki apk)
- **intermediates/javac** – skompilowane klasy (pliki class)
- **intermediates/dex** – klasy skonwertowane z formatu class do dex
- **intermediates/processed\_res** – przetworzone zasoby
- **biblioteki**

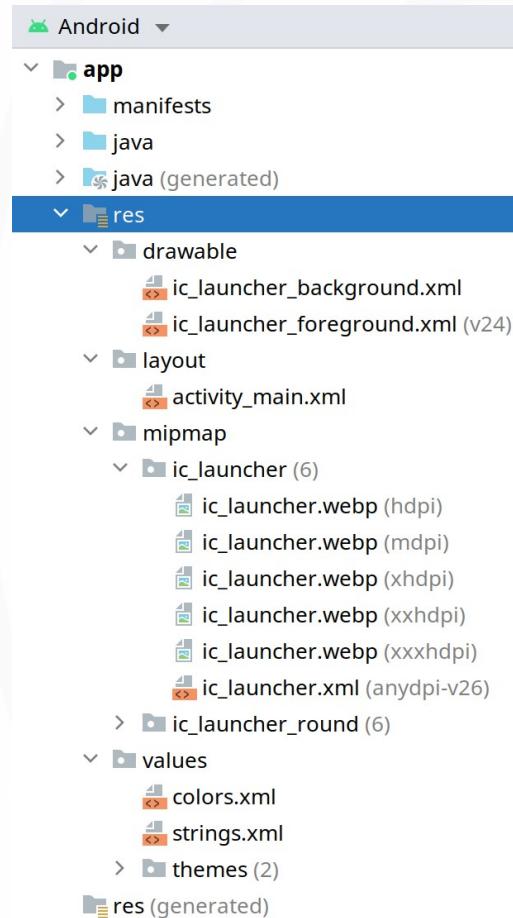
# Struktura projektu – zasoby



- res – katalog zasobów
- drawable, mipmap – elementy graficzne
- mdpi, hdpi, xhdpi... – kwalifikatory określające gęstość pikseli
- menu – wygląd menu (w obecnych wersjach *Android Studio* trzeba samodzielnie utworzyć)

# Struktura projektu – zasoby

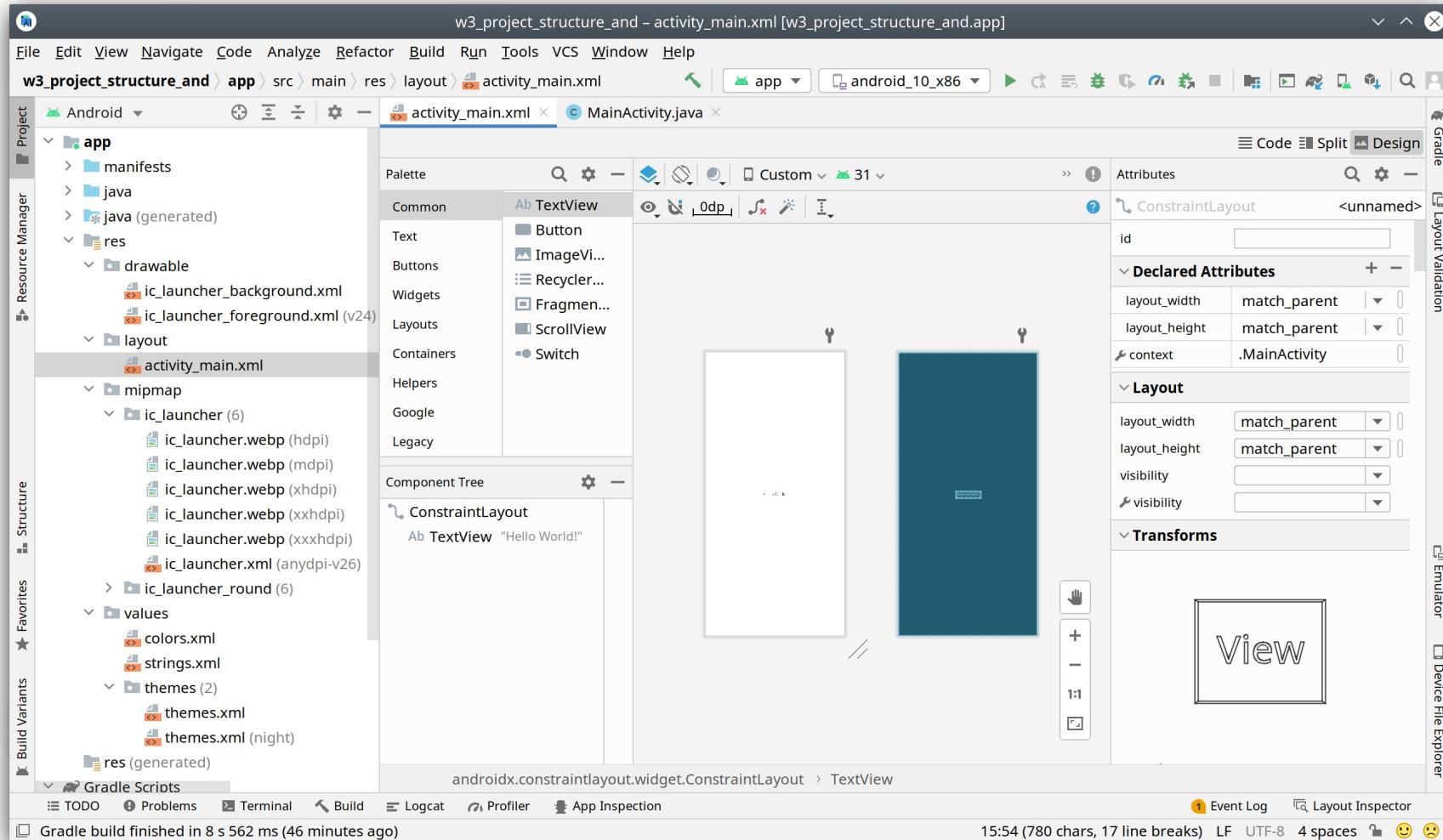
- layout – układ komponentów graficznych



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf=
            "parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

# Graficzny edytor układu



# Struktura projektu – zasoby

- values – pliki XML definiujące wartości różnych rodzajów

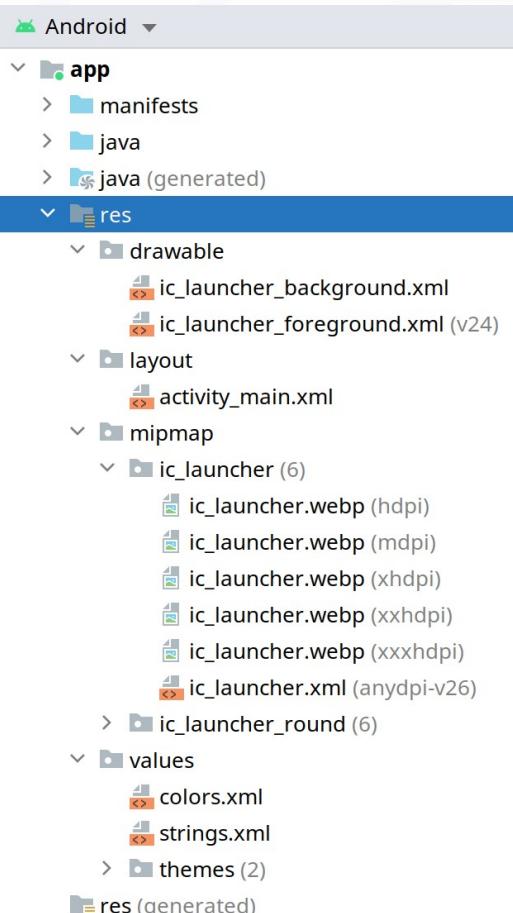
- colors.xml – kolory

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFF</color>
</resources>
```

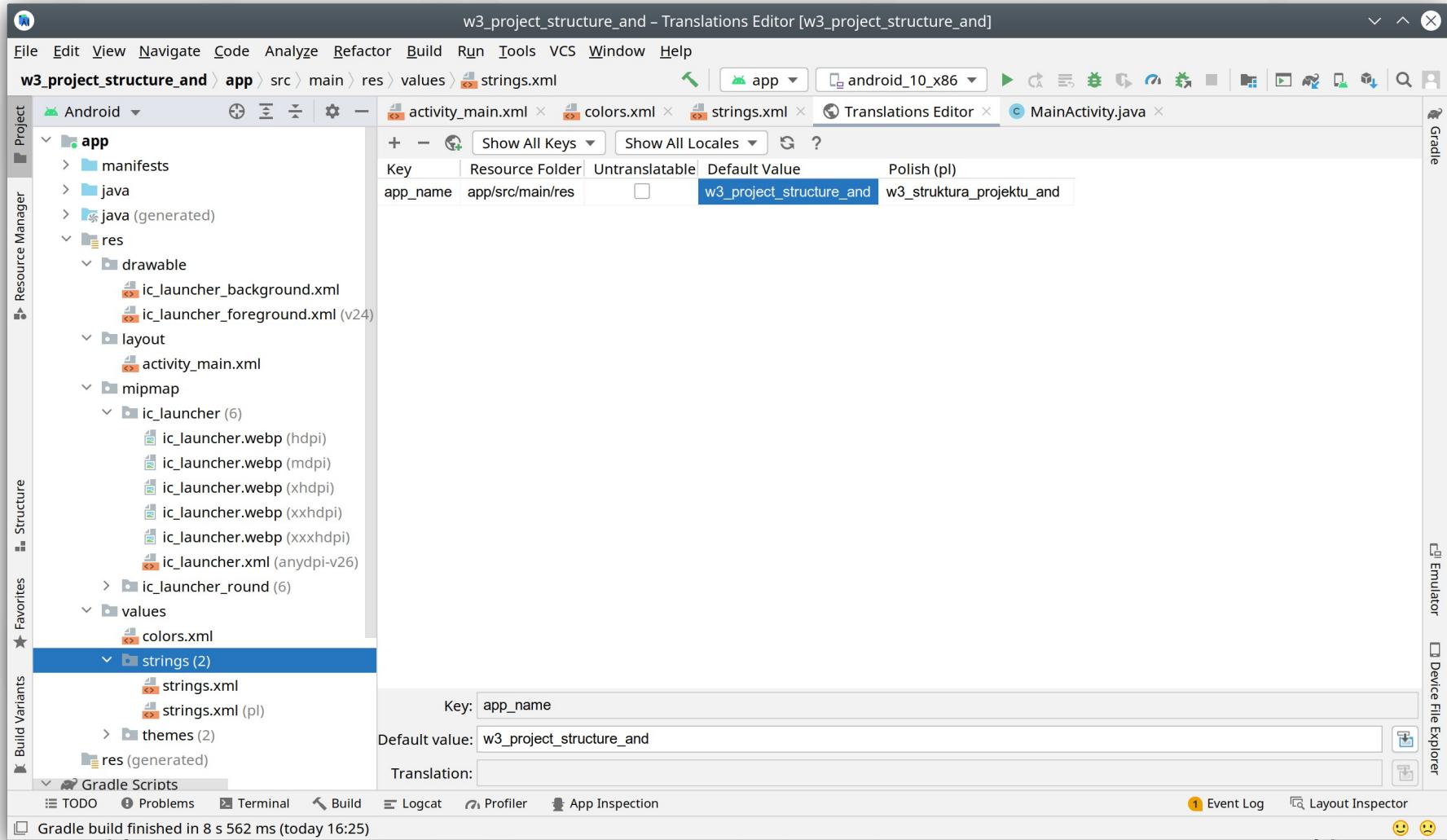
- strings.xml – teksty wyświetlane przez aplikację

```
<resources>
    <string name="app_name">
        w3_project_structure_and</string>
</resources>
```

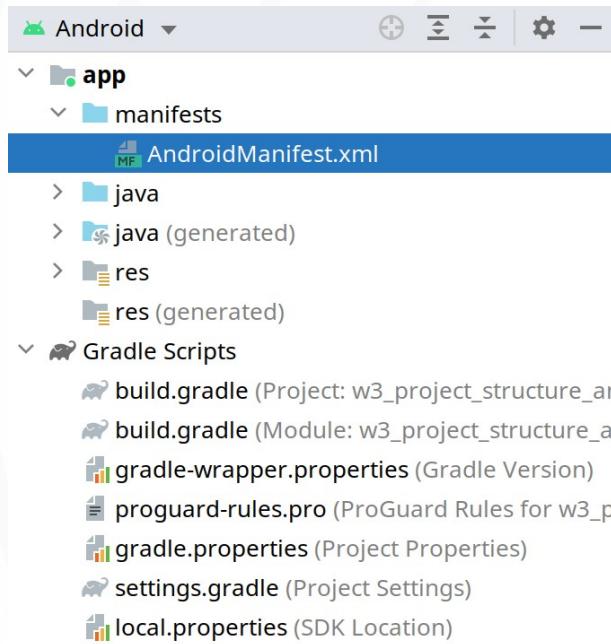
- dimens.xml – rozmiary elementów (tworzony opcjonalnie)
- themes – motywy definiujące wygląd aplikacji



# Edytor tłumaczeń



# Struktura projektu – manifest aplikacji



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=
    "http://schemas.android.com/apk/res/android"
    package="com.example.w3_project_structure_and">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.W3_project_structure_and">

        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name=
                    "android.intent.action.MAIN" />
                <category android:name=
                    "android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

# Struktura projektu – build.gradle (app)

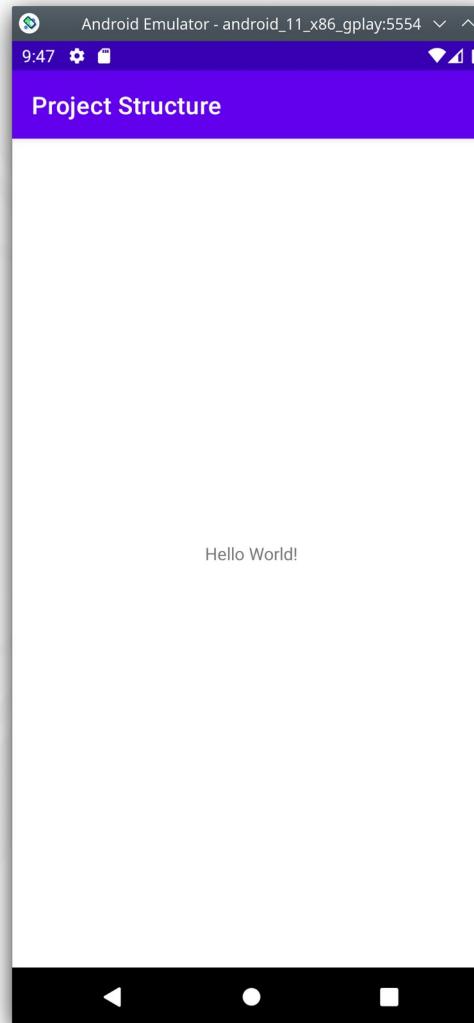
```
plugins {
    id 'com.android.application'
}

android {
    compileSdk 31
    defaultConfig {
        applicationId "com.example.w3_project_structure_and"
        minSdk 28
        targetSdk 31
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner
            "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile(
                'proguard-android-optimize.txt'),
                'proguard-rules.pro'
        }
    }
}
```

# Struktura projektu – build.gradle (app)

```
compileOptions {  
    sourceCompatibility JavaVersion.VERSION_1_8  
    targetCompatibility JavaVersion.VERSION_1_8  
}  
  
dependencies {  
    implementation 'androidx.appcompat:appcompat:1.4.0'  
    implementation 'com.google.android.material:material:1.4.0'  
    implementation 'androidx.constraintlayout:constraintlayout:2.1.2'  
    testImplementation 'junit:junit:4.+'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'  
    androidTestImplementation  
        'androidx.test.espresso:espresso-core:3.4.0'  
}
```

# Działająca aplikacja



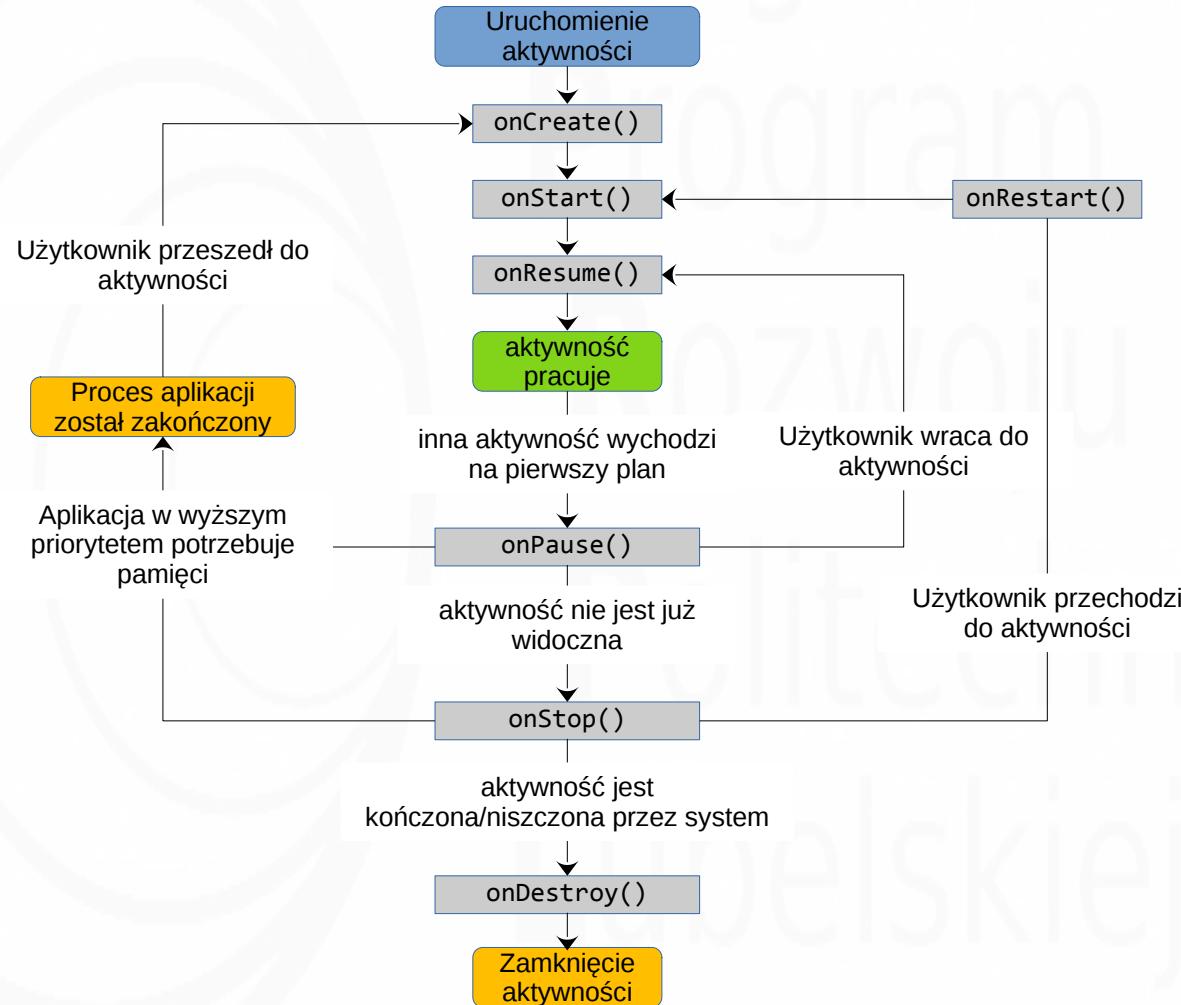
# Aktywności – podstawowy składnik aplikacji

- Aktywność (ang. *Activity*) – podstawowy składnik aplikacji dla systemu Android
- Reprezentuje czynność, którą może wykonać użytkownik
- Wywoływane przez użytkownika (za pomocą *launcher*) lub przez inne aktywności (nawet w innych aplikacjach)
- Istotnym pojęciem jest *cykl życia aktywności*. Aktywność może być:
  - aktywna – aktywność jest widoczna na pierwszym planie, użytkownik może jej używać. Posiada fokus w trybie wielo- okienkowym

# Aktywności – podstawowy składnik aplikacji

- *wstrzymana (zapauzowana)* – aktywność jest częściowo przesłonięta przez inny element (nie otrzymuje informacji o zdarzeniach) lub nie posiada fokusu w trybie wielo-okienkowym. Powinna:
  - zatrzymać animacje i inne działania obciążające CPU
  - zachować niezapisane zmiany (tylko gdy użytkownik oczekuje, że zostaną zapisane na stałe)
  - zwolnić zasoby takie jak odbiorcy komunikatów, sensory (np. GPS, aparat) itp.
- *zatrzymana* – całkowicie niewidoczna (w tle) lub utraciła fokus w trybie wielo-okienkowym
  - powinna zwolnić wszystkie zasoby, które nie są potrzebne
  - może zostać zniszczona przez system gdy potrzebuje system pamięci

# Aktywności – uproszczony cykl życia



źródło: <https://developer.android.com/guide/components/activities/activity-lifecycle>



## Programowanie Aplikacji Mobilnych na Platformę Android

**Wykład 4 – Tworzenie graficznego interfejsu  
użytkownika. Sposoby przekazywania danych  
pomiędzy elementami aplikacji.**

Jakub Smołka



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



# Elementy interfejsu użytkownika

- Komponenty graficzne często nazywane widokami reprezentowane są przez dwie podstawowe kategorie obiektów - obiekty klas dziedziczących po View i ViewGroup
  - Klasy pochodne View to komponenty, które rysują na ekranie i z którymi użytkownik może wchodzić w interakcje
  - Klasy pochodne ViewGroup to pojemniki na: inne pojemniki i zwykłe widoki
  - Klasa ViewGroup dziedziczy po View
- Pochodne ViewGroup: RadioGroup (grupa przycisków radiowych), LinearLayout (układ liniowy elementów), RecyclerView (lista/siatka elementów), TableLayout (układ tabelaryczny) ...
- Pochodne View: TextView (etykieta tekstowa), EditText (pole tekstowe), Button (przycisk), CheckBox (pole wyboru), RadioButton (przycisk radiowy) ...

# Sposoby tworzenia interfejsu użytkownika

- *Podejście deklaratywne* – deklarowanie elementów interfejsu w plikach XML
  - Obiekty zostaną stworzone na podstawie XML w momencie wywołania metody `setContentView(id)` w aktywności
  - Nazwy elementów w XML odpowiadają nazwom klas np. `<TextView>` ↔ `TextView`, `<LinearLayout>` ↔ `LinearLayout`
  - Pozwala na oddzielenie prezentacji i logiki. Ułatwia to uwzględnienie różnych rodzajów urządzeń
- *Podejście programowe* – stworzenie kodu, który tworzy widoki i budowanie hierarchii komponentów w kodzie Java
- Oba podejście można stosować w zależności od potrzeb np. stworzyć układ w XML a następnie modyfikować go dynamicznie w trakcie wykonania aplikacji

# Podstawowi zarządcy układu

- Szczególnie ważna kategoria pochodnych ViewGroup
- W aplikacjach mobilnych dla systemu Android szczególnie ważne jest tworzenie responsywnych układów dostosowujących się do urządzenia
- *Liniowy* – LinearLayout
  - rozmieszcza elementy w jednej kolumnie lub jednym wierszu
- *Z ograniczeniami* – ConstraintLayout
  - zaawansowany układ, który rozmieszcza elementy w oparciu o narzucone ograniczenia
- *Tabelaryczny* – TableLayout
  - zachowanie podobne do tabel w HTML
  - rozmieszcza elementy w tabeli

# Podstawowe atrybuty widoków

- Elementy posiadają atrybuty. Najbardziej typowe to:
  - `android:id="@+id/unikalnyIdentyfikator"` definiuje unikalny identyfikator elementu
  - `android:text="@string/unikalnyIdentyfikator"` określa tekst etykiety lub początkową wartość pola tekstopowego (zalecane odwołanie do zasobów)
  - `android:orientation="vertical"` lub `"horizontal"` pozwala (m.in.) na określenie sposobu rozмещения элементов przez liniowego zarządcę układu
  - `android:layout_width="match_parent"` lub `"wrap_content"` i `android:layout_height="match_parent"` lub `"wrap_content"` pozwalają na ustalenie szerokości oraz wysokości zarządcy/komponentu

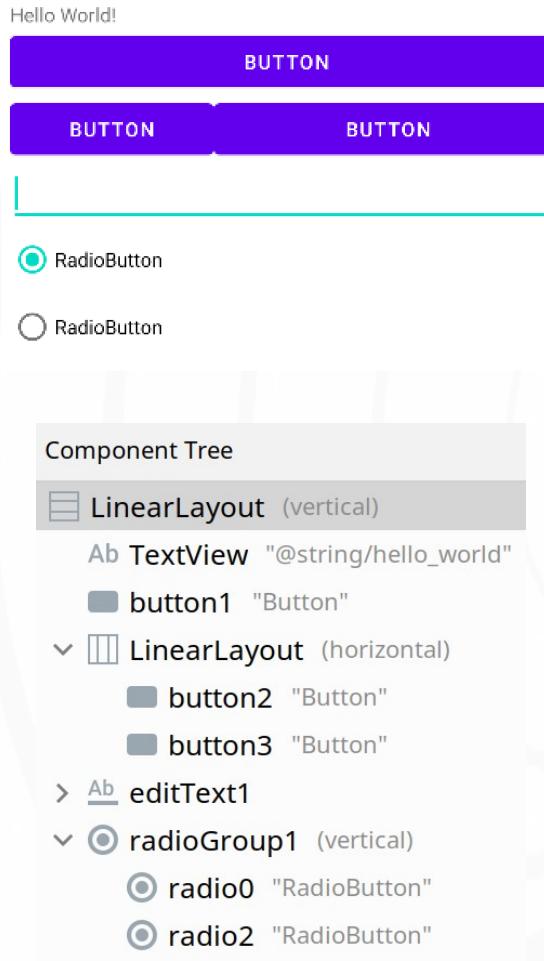
# Podstawowe atrybuty widoków

- Wartość `match_parent` powoduje, że zarządcą/komponent wypełni cały obszar zajmowany przez swojego rodzica
- Wartość `wrap_content` powoduje, że rozmiar komponentu zostanie dopasowany do zawartości
- `android:paddingBottom = "@dimen/activity_vertical_margin"`,
- `android:paddingStart = "@dimen/activity_horizontal_margin"`,
- `android:paddingEnd = "@dimen/activity_horizontal_margin"`,
- `android:paddingTop = "@dimen/activity_vertical_margin"`
- określają odpowiednio dolny, lewy, prawy i górny margines wewnętrzny. Są podobne do atrybutu `margin`, który określa margines zewnętrzny
- atrybuty `...Start...` i `...End...` są przykładem atrybutów, które zależą od ustawień językowych

# Podstawowe atrybuty widoków

- `android:layout_weight="?"` - określa, w jakim stopniu komponenty są rozciągane przez liniowego zarządcę układu (LinearLayout)
  - 0 - komponent nie jest rozciągany
  - w innym wypadku piksele dostępne w układzie będą rozdysponowane proporcjonalnie do wartości atrybutu (np. gdy `layout_weight` jest równe 2 i 3, to pierwszy otrzyma 2/5 a drugi 3/5 wolnej przestrzeni)
- `android:weight_sum=?` - ustawia sumę wag
- `android:visibility="gone"` lub `"invisible"` lub `"visible"` pozwala na określenie widoczności elementów
  - `visible` - element jest widoczny
  - `gone` – element jest niewidoczny (nie ma pustego miejsca)
  - `invisible` – element jest niewidoczny (widoczne jest puste miejsce)

# LinearLayout – przykład



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
    android:orientation="vertical"
    android:padding="@dimen/activity_padding"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button" />

        <Button
            android:id="@+id/button3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button" />
    </LinearLayout>

    <RadioGroup
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <RadioButton
            android:id="@+id/radio0"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <RadioButton
            android:id="@+id/radio2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </RadioGroup>
</LinearLayout>
```

# LinearLayout – przykład

Hello World!

BUTTON  
BUTTON

RadioButton

RadioButton

Component Tree

LinearLayout (vertical)  
  Ab TextView "@string/hello\_world"  
  button1 "Button"  
  LinearLayout (horizontal)  
    button2 "Button"  
    button3 "Button"  
  editText1  
  radioGroup1 (vertical)  
    radio0 "RadioButton"  
    radio2 "RadioButton"

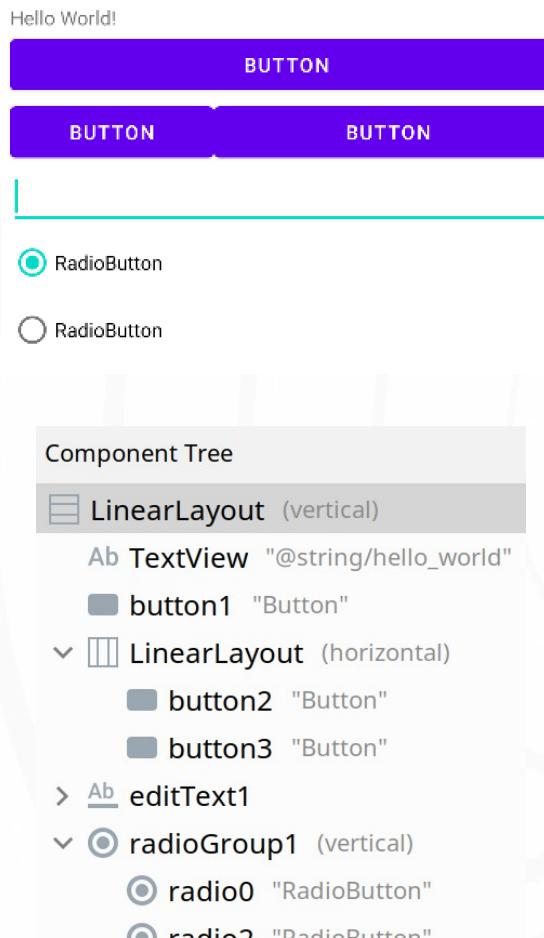
```
<LinearLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content">
```

```
<Button  
    android:id="@+id/button2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_weight="1"  
    android:text="Button" />
```

```
<Button  
    android:id="@+id/button3"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_weight="3"  
    android:text="Button" />
```

```
</LinearLayout>
```

# LinearLayout – przykład



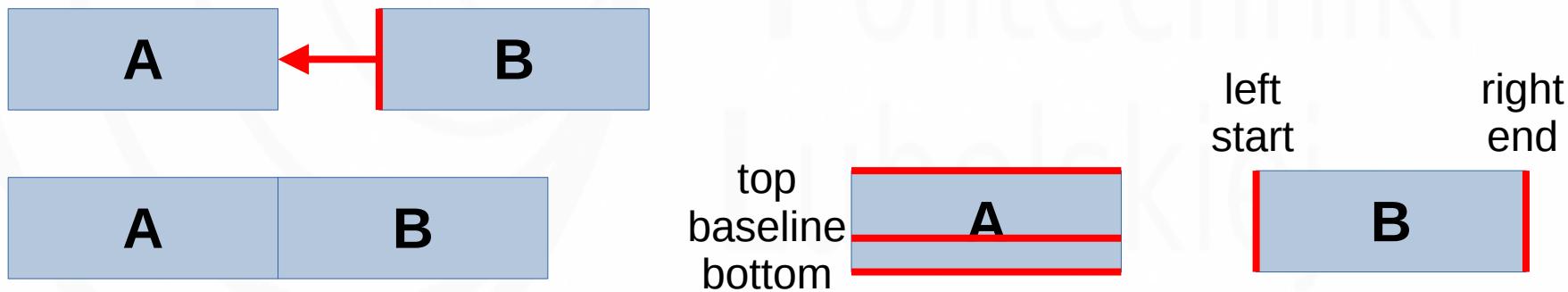
```
<EditText  
    android:id="@+id/editText1"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content">  
    <requestFocus />  
</EditText>  
  
<RadioGroup  
    android:id="@+id/radioGroup1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content">  
    <RadioButton  
        android:id="@+id/radio0"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:checked="true"  
        android:text="RadioButton" />  
    <RadioButton  
        android:id="@+id/radio2"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="RadioButton" />  
    </RadioGroup>  
</LinearLayout>
```

# ConstraintLayout

- Układ z ograniczeniami umożliwia:
- *Rozmieszczenie względne* – określanie położenia komponentu względem innego: wyrównanie krawędzi, ustawianie marginesów (odstępu od innego elementu), ustawianie marginesów dla „znikniętych” (*gone*) elementów, wyrównanie linii bazowych)
- *Rozmieszczenie kołowe* – rozmieszczenie elementów na okręgu, którego środkiem jest komponent odniesienia
- Ograniczanie rozmiarów komponentów (rozmiary minimalne, maksymalne, procentowe)
- Tworzenie łańcuchów elementów (ograniczenia dwukierunkowe)

# ConstraintLayout – atrybuty

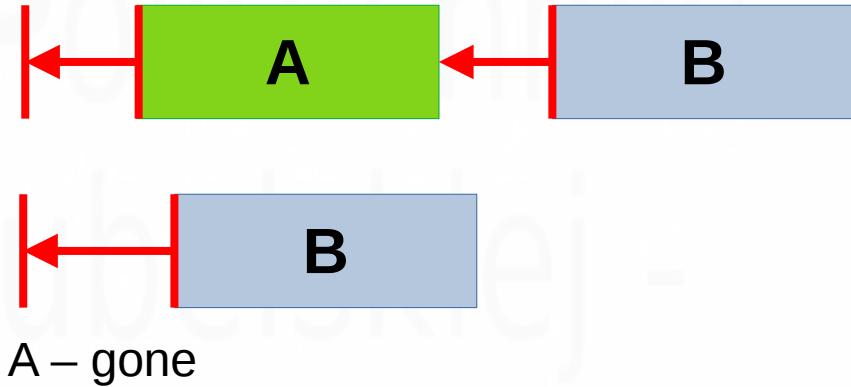
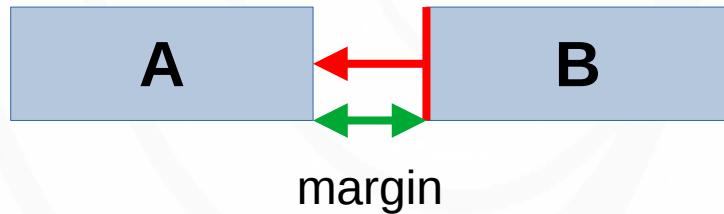
- Atrybuty powodujące, że krawędź komponentu będzie zrównana z krawędzią innego komponentu lub rodzica:  
`layout_constraintLeft_toLeftOf`,  
`layout_constraintLeft_toRightOf`, ...,  
`layout_constraintTop_toTopOf`, ...,  
`layout_constraintBottom_toBottomOf`,  
`layout_constraintBaseline_toBaselineOf`,  
`layout_constraintStart_toEndOf`, ...



źródło: <https://developer.android.com/reference/android/support/constraint/ConstraintLayout>

# ConstraintLayout – atrybuty

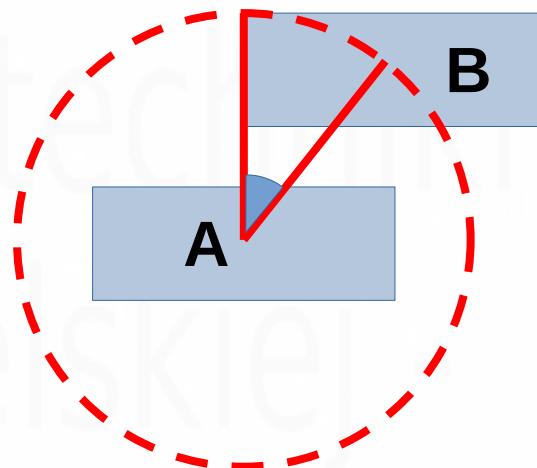
- Typowe atrybuty określające marginesy:  
`android:layout_marginStart`,  
`android:layout_marginEnd`
- Marginesy dla „znikniętych” elementów:  
`layout_goneMarginStart`, `layout_goneMarginEnd`



źródło: <https://developer.android.com/reference/android/support/constraint/ConstraintLayout>

# ConstraintLayout – atrybuty

- Określenie proporcji w jakiej dzielone jest wolne miejsce po obu stronach elementu:  
`layout_constraintHorizontal_bias`,  
`layout_constraintVertical_bias` – wartość np. 0.3
- Rozmieszczenie kołowe: `layout_constraintCircle` (określa inny komponent),  
`layout_constraintCircleRadius`,  
`layout_constraintCircleAngle`



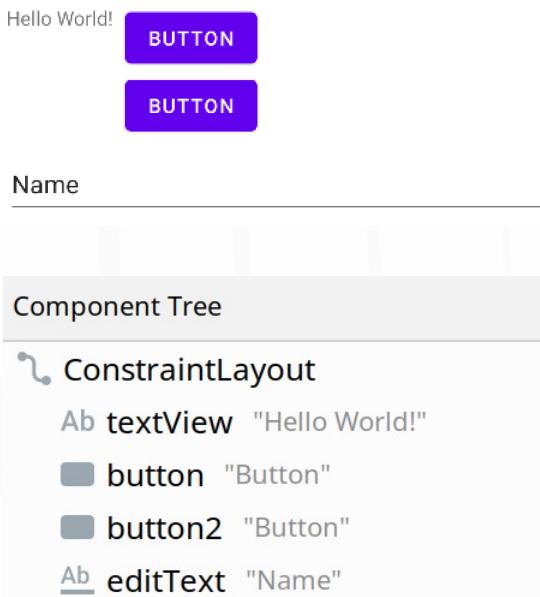
źródło: <https://developer.android.com/reference/android/support/constraint/ConstraintLayout>

# ConstraintLayout – atrybuty

- Ograniczenie rozmiarów samego layoutu:  
`android:minWidth`, `android:minHeight`,  
`android:maxWidth`, `android:maxHeight`
- Ograniczenie rozmiarów komponentu:
  - Określają minimalny/maksymalny rozmiar:  
`layout_constraintWidth_min`,  
`layout_constraintHeight_min`,  
`layout_constraintWidth_max`,  
`layout_constraintHeight_max`
  - Ustawiają rozmiar równy procentowi rozmiaru rodzica:  
`layout_constraintWidth_percent`,  
`layout_constraintHeight_percent` (wartości [0;1])

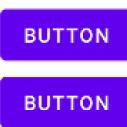
# ConstraintLayout – przykład

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="Hello World!"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:layout_marginStart="8dp"
        android:text="Button"
        app:layout_constraintStart_toEndOf="@+id/textView"
        app:layout_constraintTop_toTopOf="parent" />
```



# ConstraintLayout – przykład

Hello World!



Name

Component Tree

ConstraintLayout

  Ab textView "Hello World!"

  button "Button"

  button2 "Button"

  Ab editText "Name"

```
<Button  
    android:id="@+id/button2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Button"  
    app:layout_constraintEnd_toEndOf="@+id/button"
```

```
        app:layout_constraintStart_toStartOf="@+id/button"  
        app:layout_constraintTop_toBottomOf=  
            "@+id/button" />
```

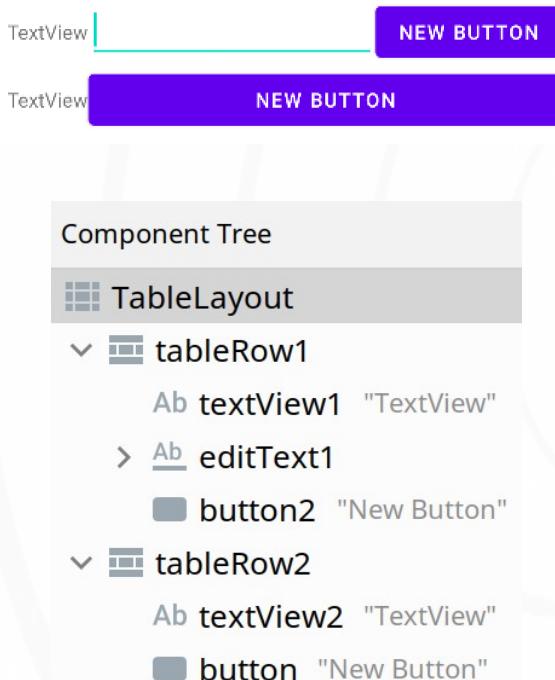
```
<EditText
```

```
    android:id="@+id/editText"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:layout_margin="8dp"  
    android:inputType="textPersonName"  
    android:minHeight="48dp"  
    android:text="Name"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf=  
        "@+id/button2" />  
</androidx.constraintlayout.widget.ConstraintLayout>
```

# TableLayout – atrybuty

- `android:shrinkColumns="*"` – określa, które kolumny mogą zostać zwężone, aby tabela zmieściła się w obszarze zadanym przez element nadzędny (kolumny wymienione po przecinku (numerowane od 0), `,"*"` – wszystkie kolumny)
- `android:stretchColumns="2"` – określa, które kolumny mogą zostać rozciągnięte, aby wypełnić wolne miejsce
- `android:layout_column="1"` – określa kolumnę, w której należy umieścić widok
- `android:layout_span="2"` – określa ile kolumn ma zajmować dany element np. pole tekstowe (domyślnie każdy komponent zajmuje tylko jedną kolumnę)

# TableLayout – przykład



```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="8dp"
    android:stretchColumns="1">
    <TableRow
        android:id="@+id/tableRow1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="TextView" />
        <EditText
            android:id="@+id/editText1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:requestFocus />
    </TableRow>
    <TableRow
        android:id="@+id/tableRow2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView
            android:id="@+id/textView2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="TextView" />
        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="New Button" />
    </TableRow>
</TableLayout>
```

# TableLayout – przykład

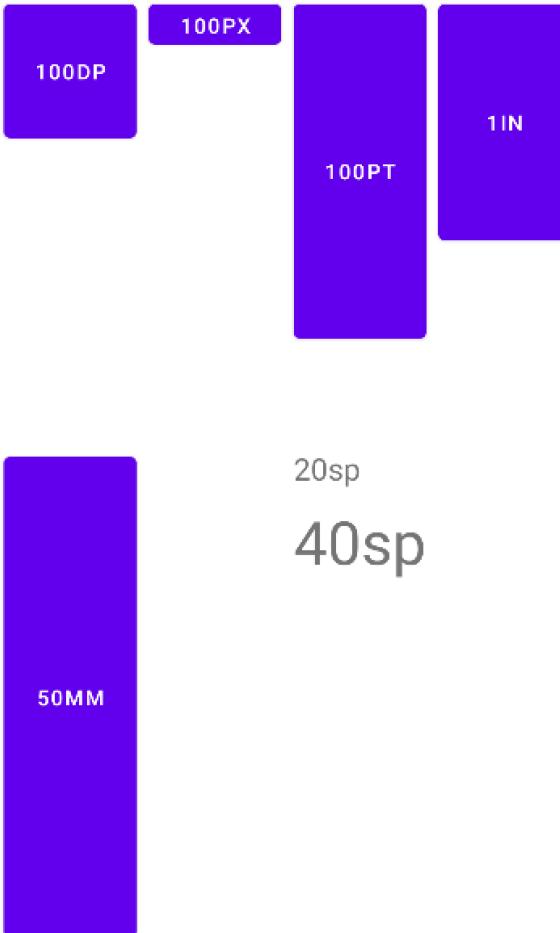


```
<Button  
    android:id="@+id/button2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_column="2"  
    android:text="New Button" />  
</TableRow>  
  
<TableRow  
    android:id="@+id/tableRow2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content">  
    <TextView  
        android:id="@+id/textView2"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="TextView" />  
    <Button  
        android:id="@+id/button"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_column="1"  
        android:layout_span="2"  
        android:text="New Button" />  
</TableRow>  
</TableLayout>
```

# Odległości

- Zalecane jest stosowanie układów elementów, które się skalują (duża różnorodność urządzeń). Czasami trzeba podać konkretne wymiary. Można stosować następujące jednostki:
  - dp (=dip) – *density-independent pixels/device-independent pixels* (na ekranie 160dpi  $1dp \approx 1px$ )
  - sp – jak dp dla rozmiarów czcionek (uwzględniają ustawienia rozmiaru czcionki)
  - pt – punkt 1/72 cala na ekranie
  - px – piksel
  - mm – milimetr na ekranie
  - in – cale na ekranie

# Odległości



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app=
        "http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="100dp"
        android:layout_margin="8dp"
        android:text="100dp"
        app:layout_constraintStart_toStartOf=
            "parent"
        app:layout_constraintTop_toTopOf=
            "parent" />
    <!-- ... -->
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf=
            "@+id/button5"
        app:layout_constraintStart_toStartOf=
            "@+id/button3"
        android:text="20sp"
        android:textSize="20sp" />
    <!-- ... -->
</androidx.constraintlayout.widget.ConstraintLayout>
```

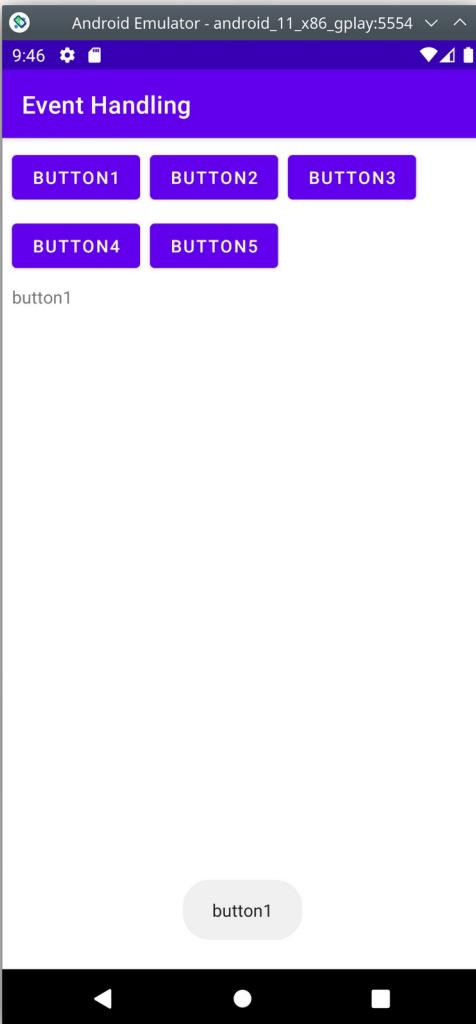
# Odwołania do widoków

- Na podstawie plików XML generowana jest klasa R
- Odwołania do elementów mają postać:  
R.rodzaj.nazwa (są to wartości typu int)  
np. R.layout.main\_activity, R.id.button1
- Aby uzyskać referencję do komponentu należy posłużyć się metodą findViewById() np.  
`Button b1 = findViewById(R.id.button1);`
- Trzeba pamiętać, że metoda zwraca <T extends View> T jest automatycznie rzutowany na typ klasy docelowej.

# Obsługa zdarzeń

- Typowo dla Javy wykorzystuje obiekty *słuchaczy* (ang. *listener*), które muszą zaimplementować odpowiedni interfejs
- Dodawane metodami `set???Listener()` lub `add???Listener()`.
- Interfejsy często są zdefiniowane wewnątrz innych klas np. `View.OnClickListener`
- Sposoby obsługi zdarzeń (wybrane)
  - Klasa anonimowa z obsługą zdarzenia
  - Wyrażenia lambda zamiast klasy anonimowej
  - Wyrażenie lambda / klasa anonimowa + wywołanie metody pomocniczej
  - XML + metoda (tylko `onClick`)
    - Musi być publiczna
    - Musi mieć parametr klasy `View`

# Obsługa zdarzeń – przykład



```
public class MainActivity extends  
    AppCompatActivity {  
    TextView textView;  
    @Override  
    protected void onCreate(  
        Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        textView = findViewById(R.id.textView);  
  
        //...  
        Button b1 = findViewById(R.id.button1);  
        b1.setOnClickListener(  
            new View.OnClickListener() {  
                @Override  
                public void onClick(View view) {  
                    textView.setText("button1");  
                    Toast.makeText(  
                        MainActivity.this, "button1",  
                        Toast.LENGTH_SHORT).show();  
                }  
            });  
    }  
}
```

# Obsługa zdarzeń – przykład

```
//...
Button b2 = findViewById(R.id.button2);
b2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        onButton2Click();
    }
});
//...
}

private void onButton2Click() {
    textView.setText("button2");
    Toast.makeText(MainActivity.this,
        "button2", Toast.LENGTH_SHORT).show();
}
```

# Obsługa zdarzeń – przykład

```
//wyrażenie Lambda zamiast interfejsu funkcyjnego
Button b3 = findViewById(R.id.button3);
b3.setOnClickListener(view -> {
    textView.setText("button3");
    Toast.makeText(MainActivity.this, "button3",
        Toast.LENGTH_SHORT).show();
});

//referencja do metody - można nią zastąpić wyrażenie Lambda
Button b4 = findViewById(R.id.button4);
b4.setOnClickListener(this::onButton4Click);
}

private void onButton4Click(View view) {
    textView.setText("button4");
    Toast.makeText(MainActivity.this,
        "button4", Toast.LENGTH_SHORT).show();
}
```

# Obsługa zdarzeń – przykład

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    <!-- ... -->

    <Button
        android:id="@+id/button5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:onClick="onButton5Click"
        android:text="Button5"
        app:layout_constraintStart_toEndOf="@+id/button4"
        app:layout_constraintTop_toBottomOf="@+id/button2" />

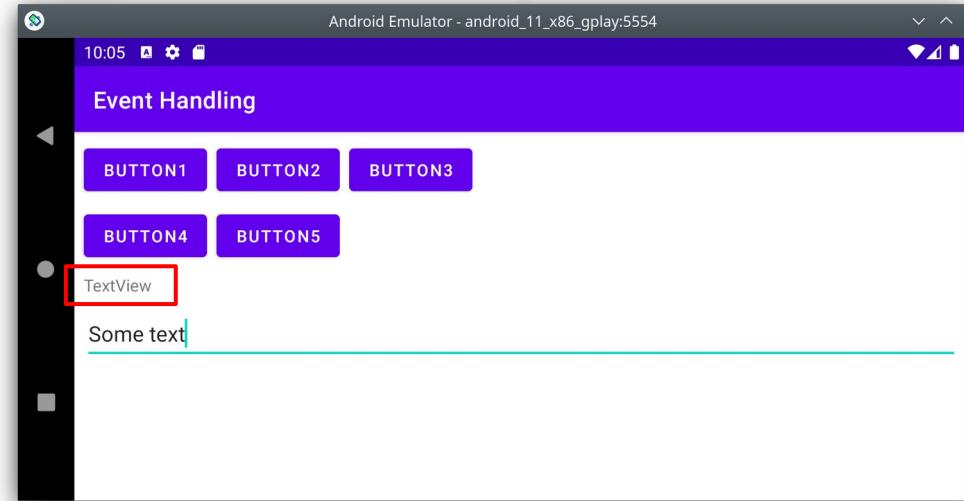
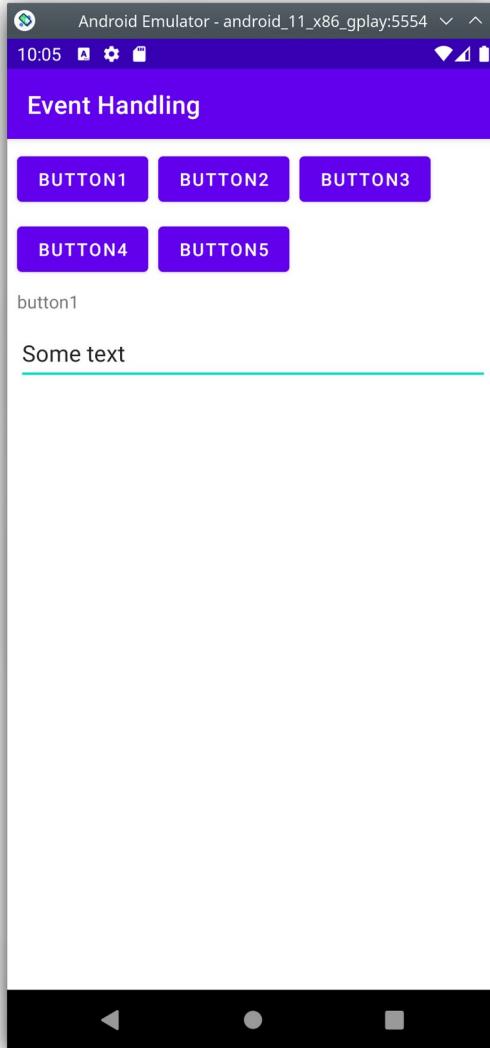
</androidx.constraintlayout.widget.ConstraintLayout>

//...
//parametrem musi być View i musi być publiczna
public void onButton5Click(View view) {
    textView.setText("button5");
    Toast.makeText(MainActivity.this, "button5",
    Toast.LENGTH_SHORT).show();
}
```

# Inne przykładowe zdarzenia

- TextWatcher – pozwala na śledzenie zmian tekstu w polu tekstowym
  - metody: `afterTextChanged()`, `beforeTextChanged()`, `onTextChanged()`
  - metoda rejestrująca: `addTextChengedListener()`
- `View.OnFocusChangeListener` – pozwala na śledzenie zmian fokusu elementów
  - metoda: `onFocusChange()`
  - metoda rejestrująca: `setOnFocusChangeListener()`
- `OnCheckedChangeListener` – pozwala na śledzenie zmian stanu zaznaczenia przycisków radiowych
  - metoda: `onClick()`
  - metoda rejestrująca: `setOnCheckedChangeListener()`

# Zachowywanie stanu aktywności



# Zachowywanie stanu aktywności

- Aktywność może zostać zniszczona i utworzona na nowo z powodu zmian w systemie (np. z powodu zmiany rozdzielczości)
- Domyślnie zachowywane są dane z obiektów View (tych które posiadają id) np. tekst w polach tekstowych EditText
- Pola aktywności i niektóre elementy (np. etykiety – TextView) nie są zachowywane
- Te informacje trzeba zachować/odtworzyć samodzielnie za pomocą metod:
  - onSaveInstanceState() / onRestoreInstanceState()
  - onSaveInstanceState() / onCreate()

# Zachowywanie stanu aktywności – przykład

```
//...

public final static String KEY_LABEL_TEXT=
    "com.example.event_handling.KEY_TEXT";

@Override
protected void onSaveInstanceState(@NonNull Bundle outState) {
    super.onSaveInstanceState(outState);
    String labelString= textView.getText().toString();
    outState.putString(KEY_LABEL_TEXT,labelString);
}

//wykonuje się po onCreate()
@Override
protected void onRestoreInstanceState(
    @NonNull Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    String labelString=
        savedInstanceState.getString(KEY_LABEL_TEXT);
    textView.setText(labelString);
}

//...
```

# Zachowywanie stanu aktywności – przykład

```
public class MainActivity extends AppCompatActivity {  
  
    TextView textView;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        textView = findViewById(R.id.textView);  
  
        //odtwarzanie stanu - 2-gi sposób  
        if (savedInstanceState!=null)  
            textView.setText(  
                savedInstanceState.getString(  
                    KEY_LABEL_TEXT));  
  
    //...  
}
```

# Wiele aktywności w aplikacji

- Aktywność skupia się na jednej czynności
- Aplikacje mogą składać się z wielu aktywności
- Aktywność może:
  - uruchomić inną aktywność i przekazać do niej dane
  - może otrzymać wyniki z uruchomionej aktywności
- Uruchamiane aktywności układane są na stosie (tzw. *back-stack*)
- Do poprzedniej aktywności można wrócić za pomocą przycisku *wstecz*
- Wszystkie aktywności muszą być zadeklarowane w manifeście. Próba uruchomienia niewymienionej w manifeście spowoduje zgłoszenie wyjątku

# Wiele aktywności – manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.w4_two_activities_and">
    <!-- ... -->
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.W4_two_activities_and">
        <activity
            android:name=".SecondActivity"
            android:parentActivityName=".MainActivity"
            android:exported="false" />
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name=
                    "android.intent.action.MAIN" />
                <category android:name=
                    "android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

# Wiele aktywności – główna aktywność, wysłanie danych

```
public class MainActivity extends AppCompatActivity {

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

        Button startSecondButton = findViewById(R.id.start_second_button);
        startSecondButton.setOnClickListener(v -> startSecondActivity());
    }

    public static final String TEXT_KEY =
        "com.example.w4_two_activities_and.text";

    private void startSecondActivity() {
        EditText text1EditText =
            findViewById(R.id.text_1_edit_text);

        Intent intent = new Intent(this, SecondActivity.class);
        intent.putExtra(TEXT_KEY,
            text1EditText.getText().toString());
        startActivity(intent);
    }
}
```

# Wiele aktywności – druga aktywność, odebranie danych

```
public class SecondActivity extends AppCompatActivity {

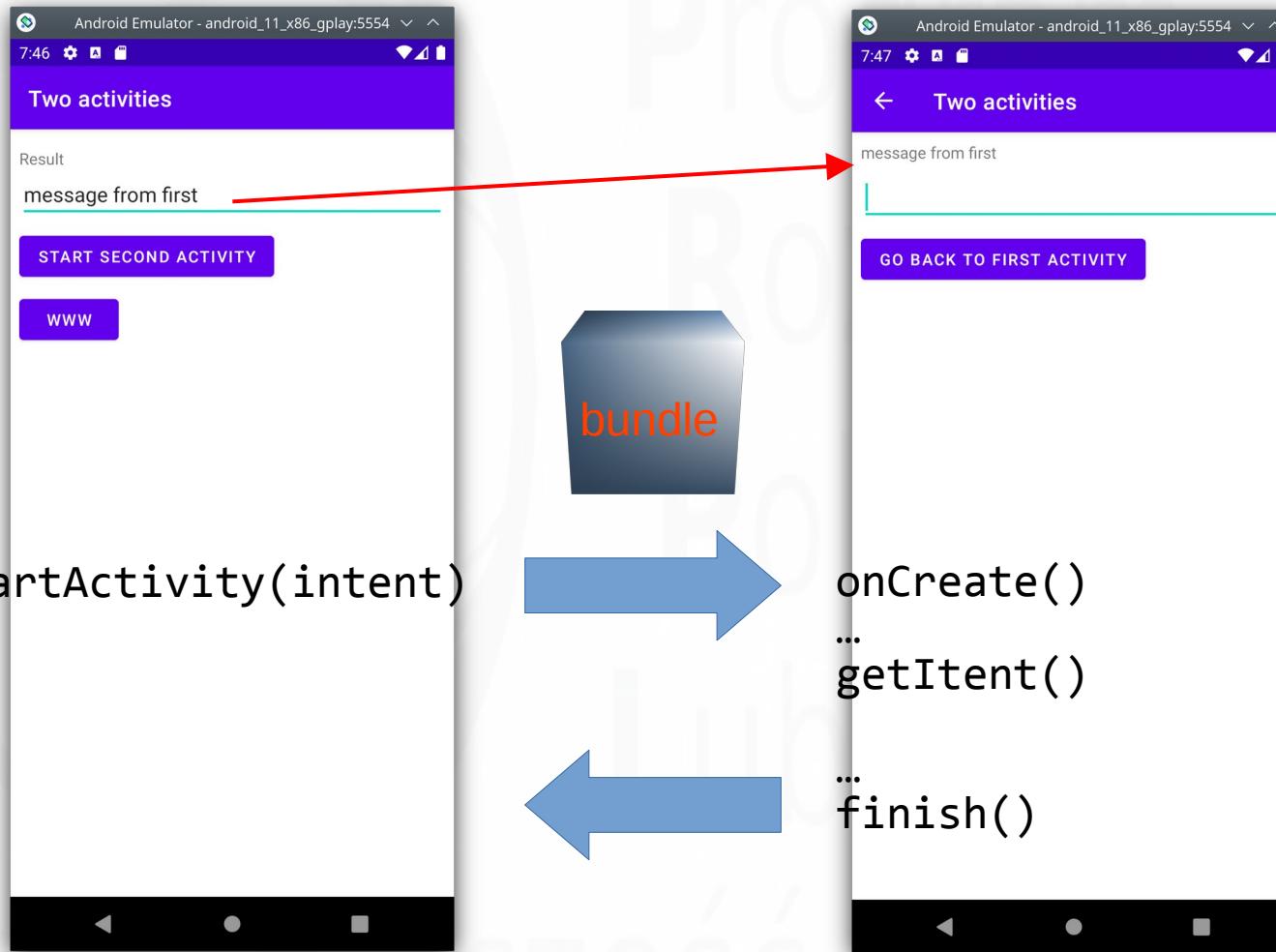
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);

        TextView textFromFirstTextView =
            findViewById(R.id.text_from_first);
        //odczytanie danych z pierwszej
        Bundle bundle = getIntent().getExtras();
        textFromFirstTextView.setText(
            bundle.getString(MainActivity.TEXT_KEY));

        Button returnButton = findViewById(R.id.return_button);
        returnButton.setOnClickListener(v -> returnToMain());
    }

    private void returnToMain() {
        finish();
    }
}
```

# Wiele aktywności – druga aktywność, odebranie danych



# Wiele aktywności – pierwsza aktywność, wywołanie zwrotne przetwarzające wynik

```
//Activity Result API - oddziela obsługę odebrania wyniku od miejsca,  
//w którym uruchamia się drugą aktywność  
  
//wynik będzie odebrany po nawet po odtworzeniu procesu (w wypadku  
//gdy został zatrzymany ze względu na brak pamięci)  
  
public class MainActivity extends AppCompatActivity {  
  
    //dodajemy pole  
    private ActivityResultLauncher<Intent> mActivityResultLauncher;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        //...  
        //rejestrujemy obsługę wyniku w momencie tworzenia  
        //aktywności; kontrakt - może też być  
        //ActivityResultContract<I,O>
```

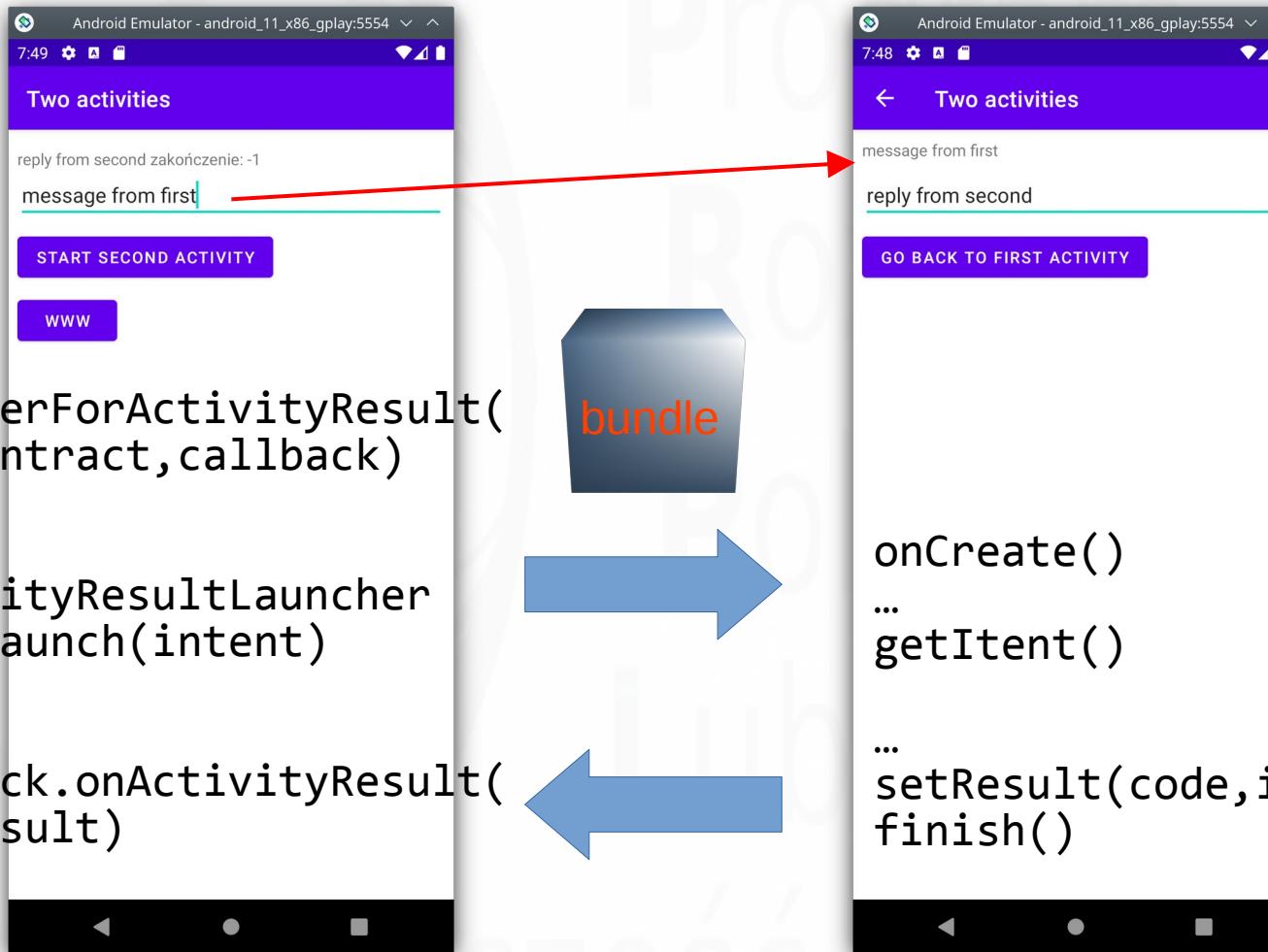
# Wiele aktywności – pierwsza aktywność, wywołanie zwrotne przetwarzające wynik

```
mActivityResultLauncher = registerForActivityResult(  
    new ActivityResultContracts  
        .StartActivityForResult(),  
        //w <> typ wyniku - tutaj ActivityResult, może  
        //też być Uri  
    new ActivityResultCallback<ActivityResult>() {  
        @Override  
        public void onActivityResult(  
            ActivityResult result) {  
            if (result.getResultCode() == RESULT_OK) {  
                Intent resultIntent = result.getData();  
                TextView resultTextView =  
                    findViewById(R.id.result_text_view);  
                resultTextView.setText(resultIntent  
                    .getStringExtra(  
                        SecondActivity.RESULT_KEY)  
                    + " zakończenie: " +  
                    result.getResultCode());  
            }  
        }  
    }  
);  
//reszta taka sama...  
}
```

# Wiele aktywności – druga aktywność, uruchomienie, zwrócenie wyników

```
private void startSecondActivity() {  
    //zamiast startActivity, tworzenie intencji identyczne...  
    mActivityResultLauncher.launch(intent);  
}  
  
public class SecondActivity extends AppCompatActivity {  
  
    //onCreate() bez zmian...  
    public final static String RESULT_KEY =  
        "com.example.w4_two_activities_and.result";  
  
    private void returnToMain() {  
        EditText text2EditText = findViewById(R.id.text_2_edit_text);  
        Intent intent = new Intent();  
        intent.putExtra(RESULT_KEY, text2EditText  
            .getText().toString());  
        setResult(RESULT_OK, intent);  
        finish();  
    }  
}
```

# Wiele aktywności – druga aktywność, odebranie danych



# Wiele aktywności – niejawna intencja

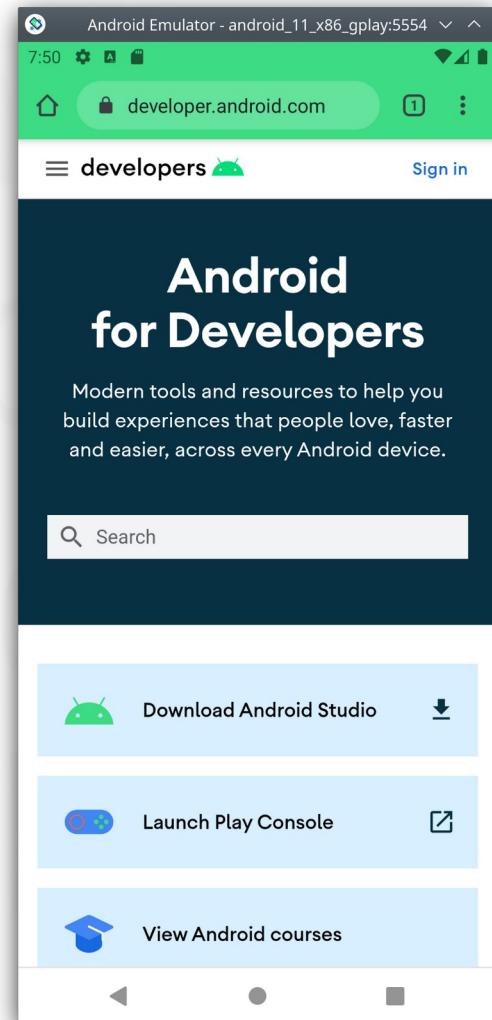
```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //reszta onCreate bez zmian...
        Button wwwButton = findViewById(R.id.www_button);
        wwwButton.setOnClickListener(v -> startWWW());
    }

    //...
    private void startWWW() {
        Intent intent = new Intent(Intent.ACTION_VIEW,
            Uri.parse("http://developer.android.com"));
        startActivity(intent);
    }
}
```

# Wiele aktywności – niejawna intencja

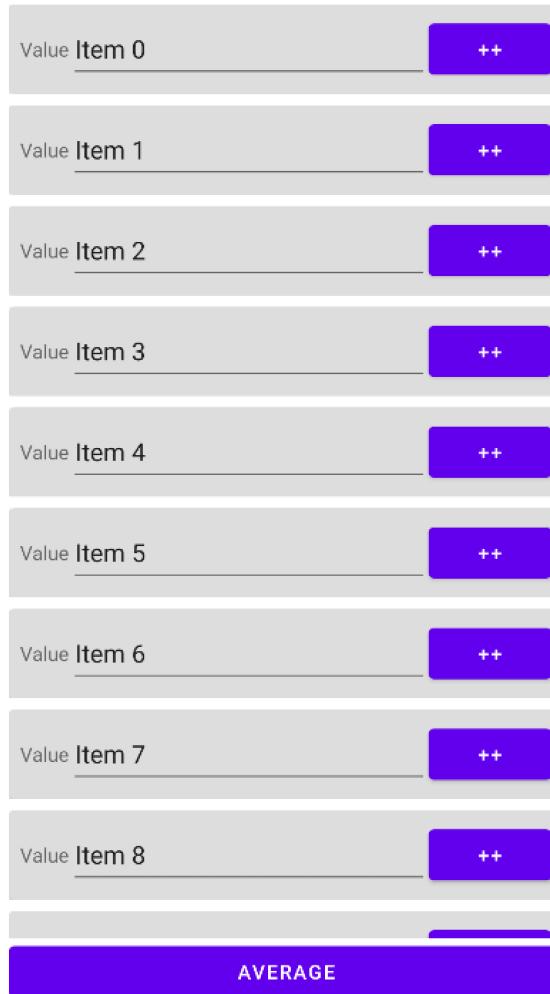
- Inne typowe akcje:
- ACTION\_VIEW – wyświetlenie
- ACTION\_SEND – wysłanie
- ACTION\_EDIT – edycja
- ACTION\_CALL – dzwonienie
- Więcej dostępnych akcji:  
<https://developer.android.com/reference/android/content/Intent>



# Listy – RecyclerView

- RecyclerView jest bardziej zaawansowaną i elastyczną wersją ListView
- Używa zarządców układu np. LinearLayoutManager, GridLayoutManager
- Elementy listy/siatki reprezentowane są przez pojemnik na widoki (RecyclerView.ViewHolder)
- Poszczególne elementy są zarządzane przez (RecyclerView.Adapter)
- Gdy zmienią się dane w źródle danych można odświeżyć widok za pomocą RecyclerView.Adapter.notify...()

# RecyclerView – layout, activity\_main.xml



```
<androidx.recyclerview.widget.RecyclerView  
    android:id="@+id/numberRecyclerView"  
    android:layout_width="0dp"  
    android:layout_height="0dp"  
    app:layout_constraintBottom_toTopOf=  
        "@+id/averageButton"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf=  
        "parent"  
    app:layout_constraintTop_toTopOf="parent"  
    tools:listitem="@layout/list_row" />
```

```
<!!-- ... -->
```

# RecyclerView – layout wiersza listy – list\_row.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingBottom="8dp"
    android:orientation="horizontal">

    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="64dp"
        app:cardBackgroundColor="@color/custom_light_gray"
        app:cardCornerRadius="4dp">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_margin="8dp">
```

# RecyclerView – layout wiersza listy – list\_row.xml

```
<TextView  
    android:id="@+id/label"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center_vertical"  
    android:text="Value" />  
<EditText  
    android:id="@+id/numberEditText"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center_vertical"  
    android:layout_weight="1"  
    android:inputType="number"  
    android:minHeight="48dp" />  
<Button  
    android:id="@+id/plusButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center_vertical"  
    android:text="++" />  
    </LinearLayout>  
</androidx.cardview.widget.CardView> Value  
</LinearLayout>
```



# RecyclerView – główna aktywność

```
public class MainActivity extends AppCompatActivity {  
    private ArrayList<Integer> mNumberList;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        Random random=new Random();  
        mNumberList=new ArrayList<>();  
        for (int i=0;i<10;i++)  
            mNumberList.add(random.nextInt(100));  
  
        Button averageButton=findViewById(R.id.averageButton);  
        averageButton.setOnClickListener(  
            view -> { computeAverage(); });
```

# RecyclerView – główna aktywność

```
RecyclerView numberRecyclerView=
    findViewById(R.id.numberRecyclerView);
MyAdapter myAdapter=new MyAdapter(this,mNumberList);
numberRecyclerView.setAdapter(myAdapter);
//RecyclerView wymaga managera układu
numberRecyclerView.setLayoutManager(
    new LinearLayoutManager(this));
}

private void computeAverage() {
    int sum=0;
    for (int number : mNumberList)
        sum+=number;
    Toast.makeText(this,"Average: "+
        (sum/10.0),Toast.LENGTH_LONG).show();
}
```

# RecyclerView – adapter

```
public class MyAdapter extends  
RecyclerView.Adapter<MyAdapter.MyAdapterViewHolder> {  
  
    private ArrayList<Integer> mNumberList;  
    private Activity mActivity;  
  
    public MyAdapter(Activity activity,ArrayList<Integer> numberList)  
{  
        mNumberList=numberList;  
        mActivity=activity;  
    }  
  
    //wywoływane gdy tworzony jest nowy wiersz  
    @NotNull  
    @Override  
    public MyAdapterViewHolder onCreateViewHolder(  
        @NotNull ViewGroup parent, int viewType) {  
        //ustawienie rodzica i attachToRoot ->  
        //wiersze będą uwzględniały rozmiary listy  
        View rowRootView=mActivity.getLayoutInflater()  
            .inflate(R.layout.list_row,parent,false);  
        MyAdapterViewHolder myAdapterViewHolder=  
            new MyAdapterViewHolder(rowRootView);  
        return myAdapterViewHolder;  
    }  
}
```

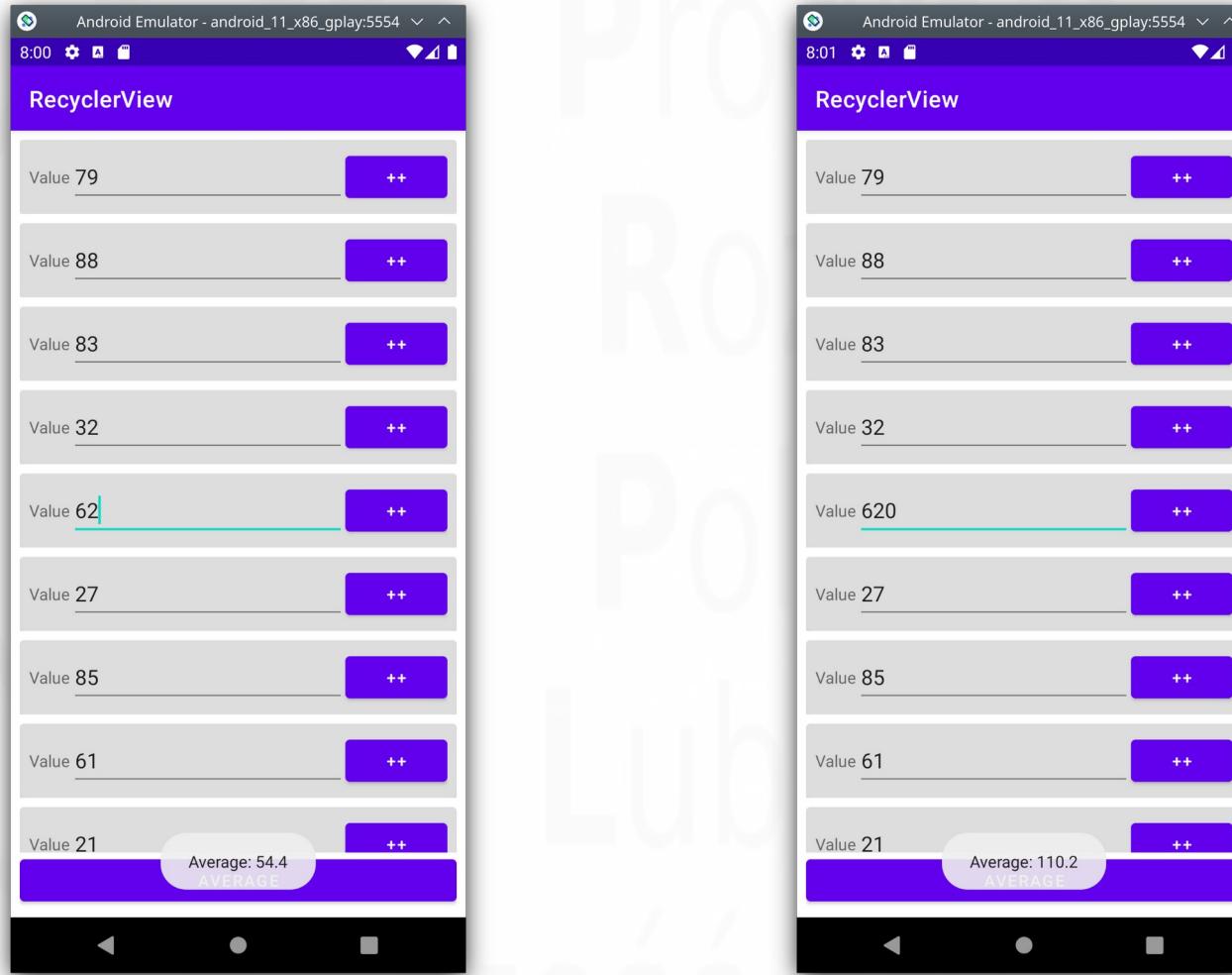
# RecyclerView – adapter

```
//wywoływany zawsze gdy ma być wyświetlony nowy wiersz
@Override
public void onBindViewHolder(@NonNull MyAdapterViewHolder holder,
    int position) {
    int value=mNumberList.get(position);
    //najpierw ustawiamy etykietę bo setText wywoła zdarzenie
    holder.mNumberEditText.setTag(position);
    holder.mNumberEditText.setText(Integer.toString(value));
}
@Override
public int getItemCount() {
    return mNumberList.size();
}
//view holder zarządza pojedynczym wierszem listy
//to dobre miejsce na zaimplementowanie słuchaczy
class MyAdapterViewHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener, TextWatcher
{
    Button mPlusButton;
    EditText mNumberEditText;
    public MyAdapterViewHolder(@NonNull View itemView) {
        super(itemView);
        mPlusButton = itemView.findViewById(R.id.plusButton);
        mPlusButton.setOnClickListener(this);
        mNumberEditText = itemView.findViewById(R.id.numberEditText);
        mNumberEditText.addTextChangedListener(this);
    }
}
```

# RecyclerView – adapter

```
@Override  
public void onClick(View v) {  
    int number=0;  
    try {  
        number=Integer.parseInt(  
            mNumberEditText.getText().toString());  
        number++;  
    } catch (NumberFormatException ignored) { }  
    mNumberEditText.setText(Integer.toString(number));  
}  
  
@Override  
public void beforeTextChanged(CharSequence s,  
    int start, int count, int after) { }  
@Override  
public void onTextChanged(CharSequence s,  
    int start, int before, int count) { }  
@Override  
public void afterTextChanged(Editable s) {  
    int number=0;  
    try {  
        number=Integer.parseInt(  
            mNumberEditText.getText().toString());  
    } catch (NumberFormatException e) { }  
    int index=(Integer) mNumberEditText.getTag();  
    mNumberList.set(index,number);  
}}}
```

# RecyclerView

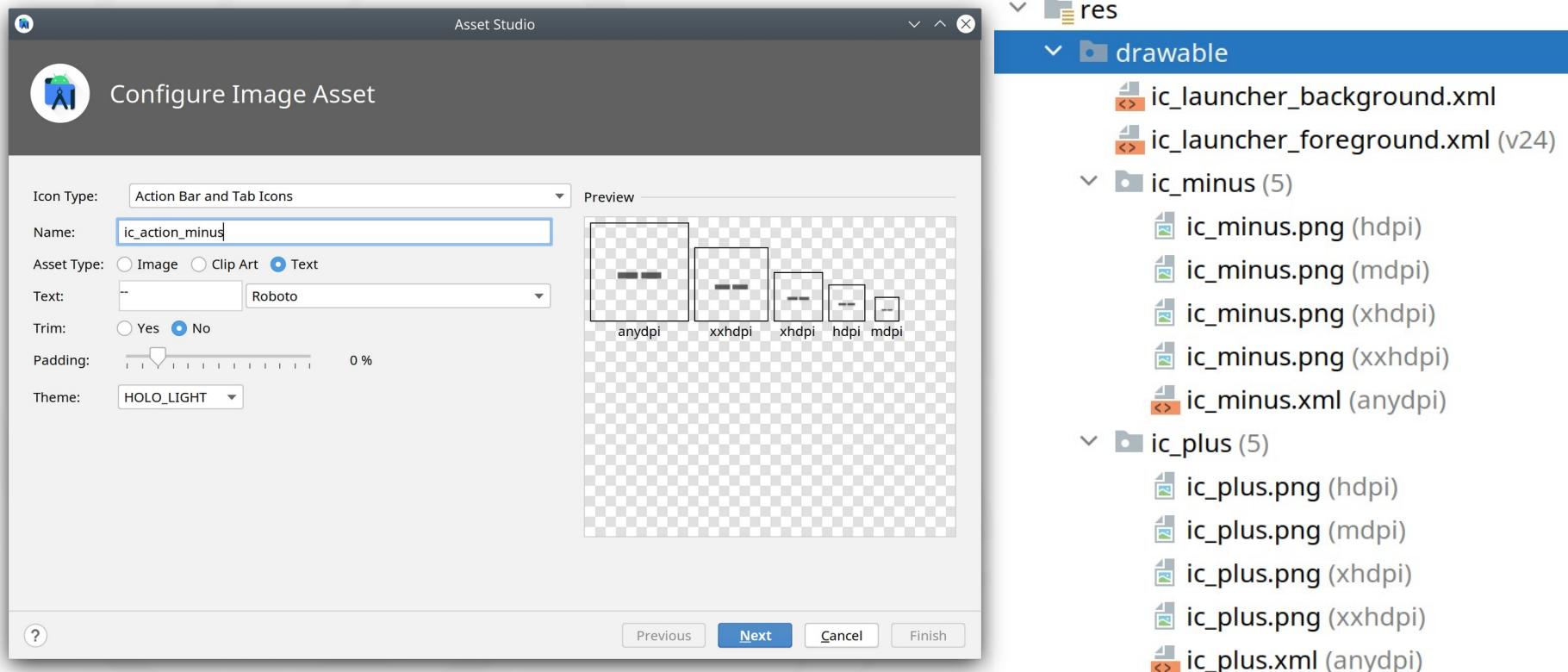


# Paski aplikacji – AppBar

- W Androidzie 2.3 i wcześniejszych menu było ukryte pod przyciskiem sprzętowym
- W Androidzie 3.0 wprowadzono pasek akcji (ActionBar) wyświetlany zazwyczaj na górze ekranu
- We współczesnych aplikacjach mogą funkcję pasków akcji przejęły paski aplikacji (AppBar)
- Klasa AppBar dostępna jest w bibliotece appcompat
- Paski aplikacji/akcji, starsze rodzaje menu są kompatybilne ze sobą (mogą różnić się sposobem prezentacji)

# Paski aplikacji – dodawanie ikon

- Ikonki dla akcji (prawy klik res | New... | Image asset)



# Paski aplikacji – typowe znaczniki/atributy

- <menu> – definiuje menu; musi być głównym znacznikiem pliku menu.xml; może być umieszczony wewnątrz znacznika <item> (submenu)
- <item> – reprezentuje element menu
- <group> – opcjonalny, niewidoczny pojemnik na opcje menu (elementy <item>); pozwala hurtowo zmieniać właściwości elementów – np. ukrywać/dezaktywować
- android:id – identyfikator opcji menu
- android:icon – ikonka
- android:title – opis akcji
- app:showAsAction - pokazywanie na pasku (never, always, ifRoom)

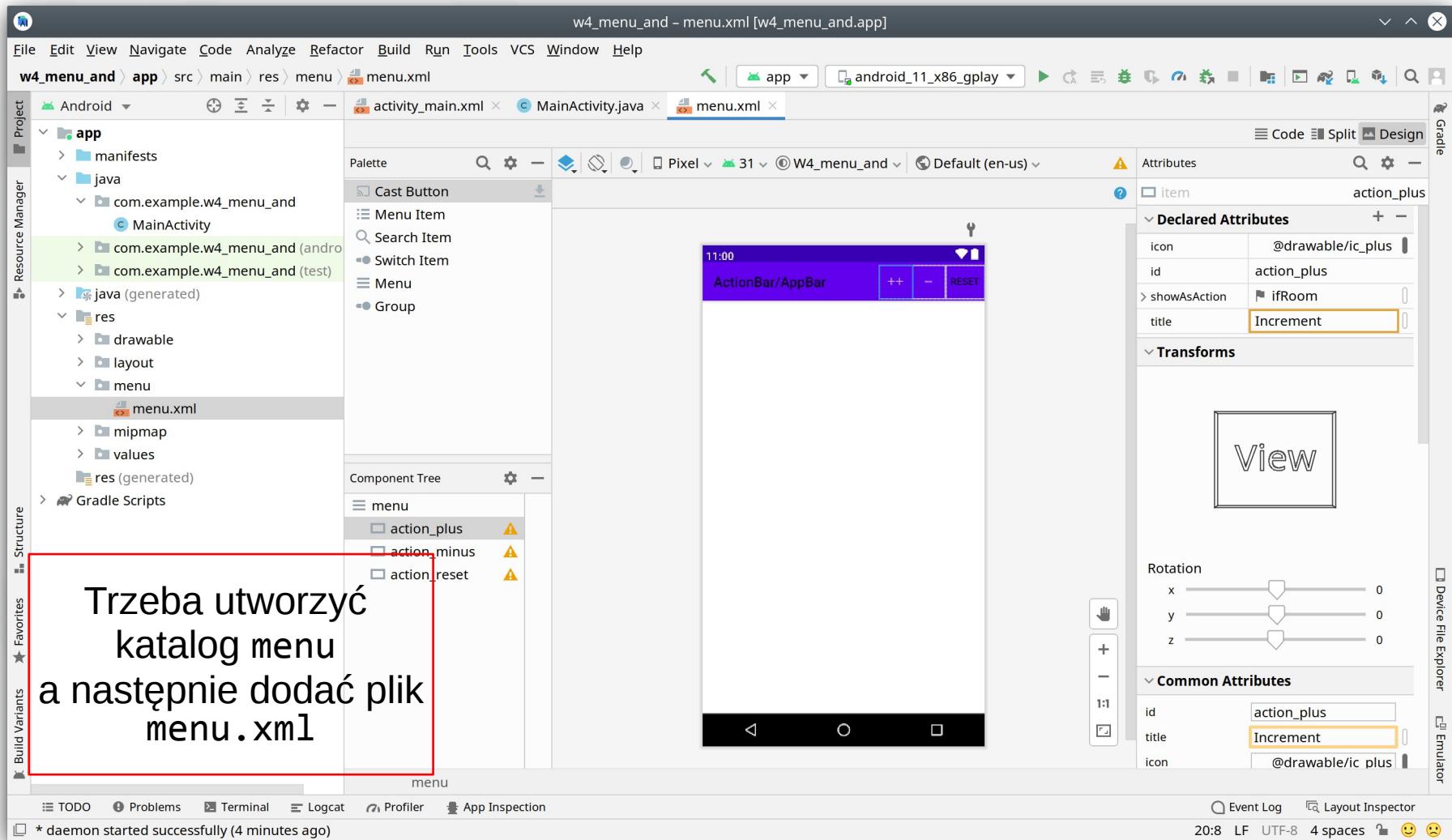
# Paski aplikacji – menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/action_plus"
        android:icon="@drawable/ic_plus"
        android:title="Increment"
        app:showAsAction="ifRoom" />
    <item
        android:id="@+id/action_minus"
        android:icon="@drawable/ic_minus"
        android:title="Decrement"
        app:showAsAction="ifRoom" />
    <item
        android:id="@+id/action_reset"
        android:title="Reset"
        app:showAsAction="ifRoom" />

</menu>
```

# Paski aplikacji – graficzny edytor paska



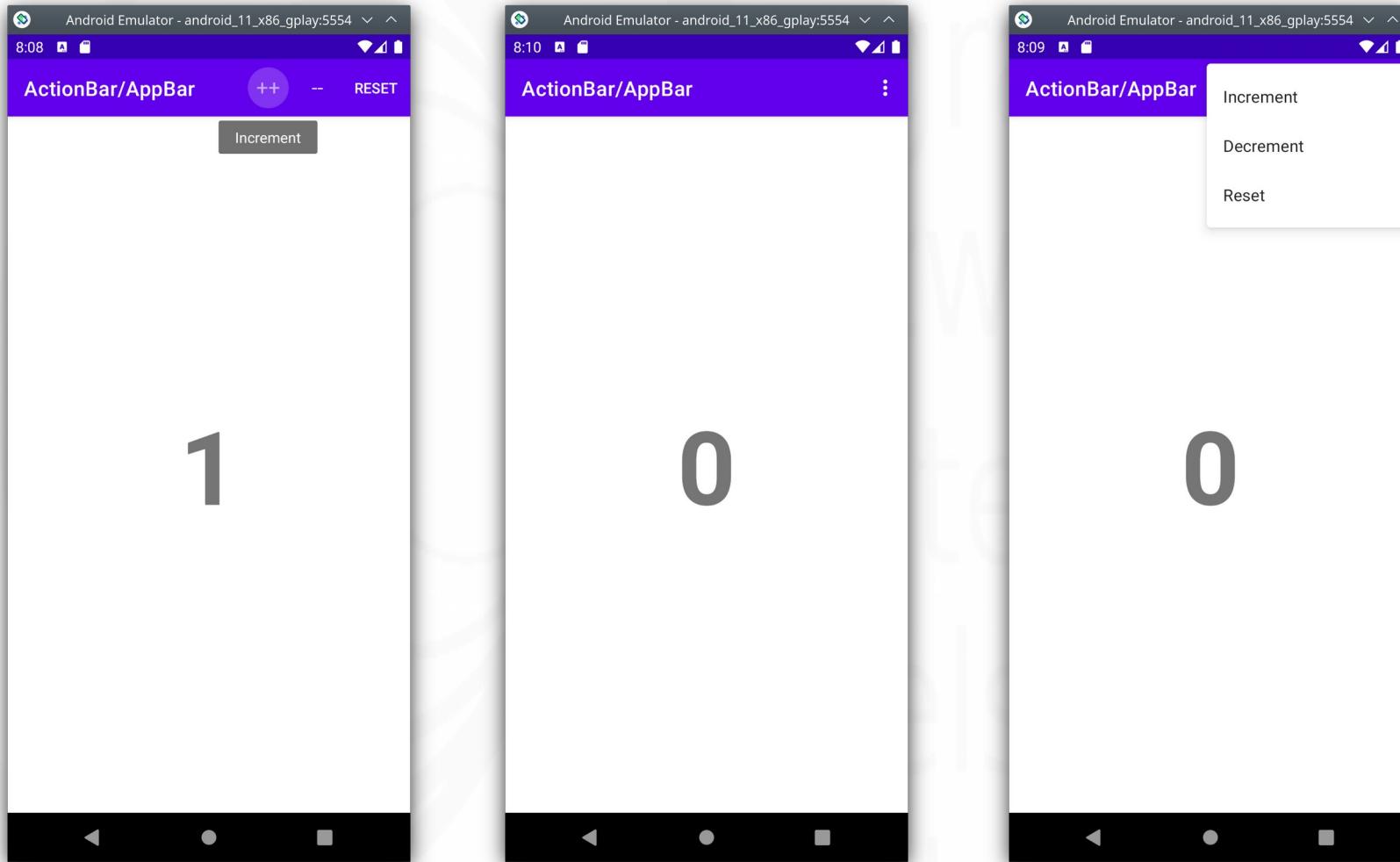
# Paski aplikacji – główna aktywność, tworzenie menu

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
  
    //wywoływana raz w momencie tworzenia menu (obecnie = tworzenia  
    //aktywności)  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) {  
        //tworzymy menu na podstawie definicji w pliku XML  
        getMenuInflater().inflate(R.menu.menu,menu);  
        return true;  
    }  
  
    //jeżeli trzeba zmodyfikować menu w za każdym razem  
    //gdy aktywność jest pokazywana należy użyć  
    //onPrepareOptionsMenu
```

# Paski aplikacji – główna aktywność, obsługa opcji menu

```
//obsługa wybranej opcji menu
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    TextView numberTv = findViewById(R.id.numberTextView);
    int number = Integer.parseInt(
        numberTv.getText().toString());
    switch (item.getItemId()) {
        case R.id.action_plus:
            numberTv.setText(Integer.toString(++number));
            break;
        case R.id.action_minus:
            numberTv.setText(Integer.toString(--number));
            break;
        case R.id.action_reset:
            numberTv.setText("0");
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

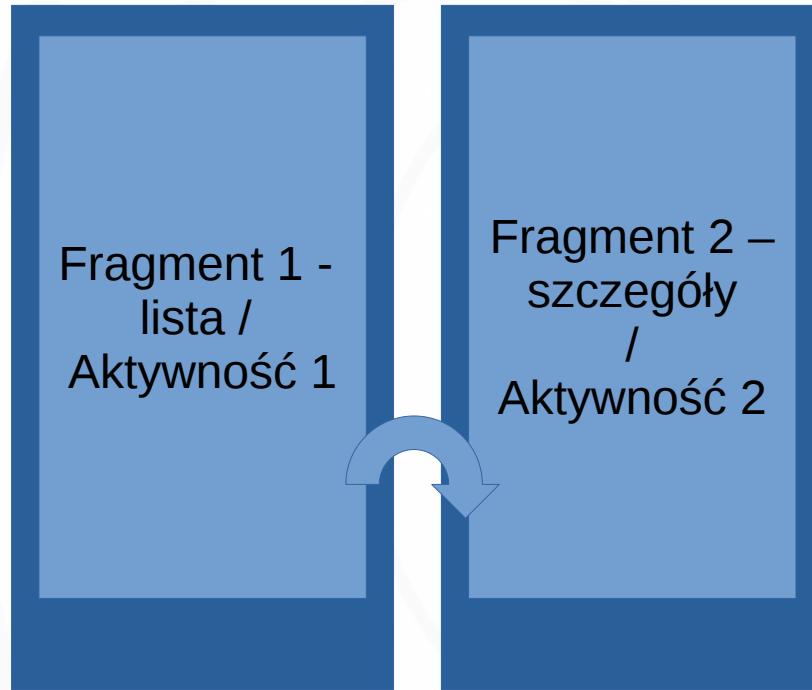
# Paski aplikacji



# Fragmenty

- Wprowadzone w Androidzie 3.0 przeznaczonym na tablety
- Ekrany większe od telefonów spowodowały problem z wyglądem aplikacji
- Powstała potrzeba tworzenia aplikacji wyglądających i działających dobrze zarówno na tabletach jak i telefonach
- Fragmenty są „mini-aktywnościami”, które można umieszczać w prawdziwych aktywnościach
- Pozwalają na tworzenie aktywności dostosowanych do różnych urządzeń ponieważ większość funkcjonalności aplikacji jest realizowana przez fragment
- Fragmenty komunikują się z aktywnością w której są osadzone
- Komunikacja między fragmentami powinna odbywać się za pośrednictwem aktywności (fragmenty powinny być uniwersalne)

# Fragmenty



Telefon

Tablet

# Fragmenty – layout, fragment\_1.xml

Fragment 1

SEND TO ACTIVITY

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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=".Fragment1">
    <TextView
        android:id="@+id/fragment1_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:text="Fragment 1"
        app:layout_constraintStart_toStartOf=
            "parent"
        app:layout_constraintTop_toTopOf=
            "parent" />
    <EditText
        android:id="@+id/fragment1_edit_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

# Fragmenty – layout, fragment\_1.xml

Fragment 1

SEND TO ACTIVITY

```
        android:layout_marginTop="8dp"
        android:layout_below=
            "@+id/fragment1_text_view"
        app:layout_constraintEnd_toEndOf=
            "parent"
        app:layout_constraintStart_toStartOf=
            "parent"
        app:layout_constraintTop_toBottomOf=
            "@+id/fragment1_text_view">
        <requestFocus />
    </EditText>
    <Button
        android:id="@+id/fragment1_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:layout_below=
            "@+id/fragment1_edit_text"
        android:text="Send to activity"
        app:layout_constraintEnd_toEndOf="parent"

        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf=
            "@+id/fragment1_edit_text" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

# Fragmenty – Fragment1.java

```
public class Fragment1 extends Fragment {  
  
    private Fragment1Listener mListener;  
  
    //interfejs do wysyłania komunikatów do aktywności  
    public interface Fragment1Listener {  
        public void messageFromFragment1(String message);  
    }  
    //wymagany pusty konstruktor  
    public Fragment1() { }  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
    }  
  
    @Override  
    public View onCreateView(LayoutInflater inflater,  
        ViewGroup container,  
        Bundle savedInstanceState) {  
        return inflater.inflate(R.layout.fragment_1, container,  
            false);  
    }  
}
```

# Fragmenty – Fragment1.java

```
//konfiguracja fragmentu
@Override
public void onViewCreated(@NonNull View rootView,
    @Nullable Bundle savedInstanceState) {
    super.onViewCreated(rootView, savedInstanceState);
    Button sendButton = rootView.findViewById(R.id.fragment1_button);
    sendButton.setOnClickListener(v -> sendToActivity());
}

//metoda pomocnicza do obsługi kliknięcia
private void sendToActivity() {
    if (mListener != null) {
        String message = ((EditText) getView()
            .findViewById(R.id.fragment1_edit_text))
            .getText().toString();
        mListener.messageFromFragment1(message);
    }
}

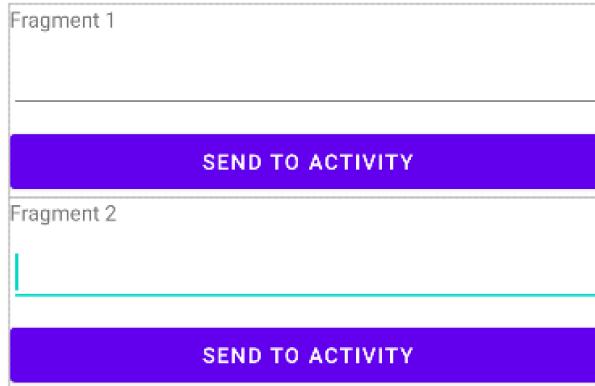
//metody publiczne do komunikacji aktywności z fragmentem
public void setText(String text) {
    //może zwrócić null jeżeli fragment nie jest widoczny
    View rootView = getView();
    if (rootView!=null) {
        EditText editText = rootView
            .findViewById(R.id.fragment1_edit_text);
        editText.setText(text);
    }
}
```

# Fragmenty – Fragment1.java

```
//podłączenie do kontekstu (np. Aktywnosci)
@Override
public void onAttach(@NonNull Context context) {
    super.onAttach(context);
    try {
        mListener = (Fragment1Listener) context;
    } catch (ClassCastException e) {
        throw new ClassCastException(context.toString()
            + " must implement OnFragmentInteractionListener");
    }
}

//odłączanie od kontekstu
@Override
public void onDetach() {
    super.onDetach();
    mListener = null;
}
```

# Fragmenty – layout, activity\_main.xml



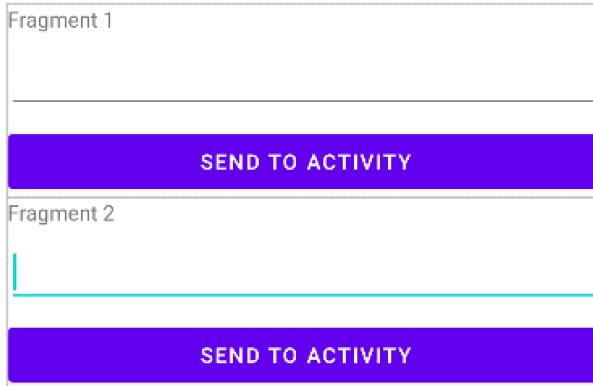
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app=
        "http://schemas.android.com/apk/res-auto"
    xmlns:tools=
        "http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="8dp"
    tools:context=".MainActivity">

    <fragment
        android:id="@+id/fragment1"
        android:name=
            "com.example.w4_fragments_and.Fragment1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf=
            "parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:layout="@layout/fragment_1" />
```

Nothing so far...  
Nothing so far...

FORWARD TO FRAGMENTS

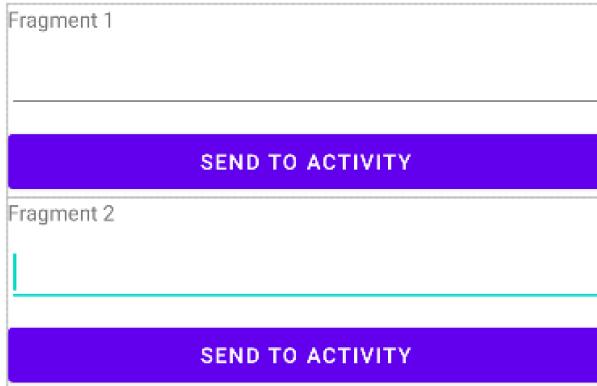
# Fragmenty – layout, activity\_main.xml



```
<fragment
    android:id="@+id/fragment2"
    android:name=
        "com.example.w4_fragments_and.Fragment2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf=
        "parent"
    app:layout_constraintTop_toBottomOf=
        "@+id/fragment1"
    tools:layout="@layout/fragment_2" />
```

```
<TextView
    android:id="@+id/text_from_1_text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:text="Nothing so far..."
    app:layout_constraintBottom_toTopOf=
        "@+id/text_from_2_text_view"
    app:layout_constraintStart_toStartOf=
        "parent" />
```

# Fragmenty – layout, activity\_main.xml



```
<TextView
```

```
    android:id="@+id/text_from_2_text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:text="Nothing so far..."
    app:layout_constraintBottom_toTopOf=
        "@+id/forward_button"
    app:layout_constraintStart_toStartOf=
        "parent" />
```

```
<Button
```

```
    android:id="@+id/forward_button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Forward to fragments"
    app:layout_constraintBottom_toBottomOf=
        "parent"
    app:layout_constraintEnd_toEndOf=
        "parent"
    app:layout_constraintStart_toStartOf=
        "parent" />
</androidx.constraintlayout
.widget.ConstraintLayout>
```

# Fragmenty – layout, activity\_main.xml (land)

- Drugi layout dla ustawienia poziomego zawiera tylko jeden fragment

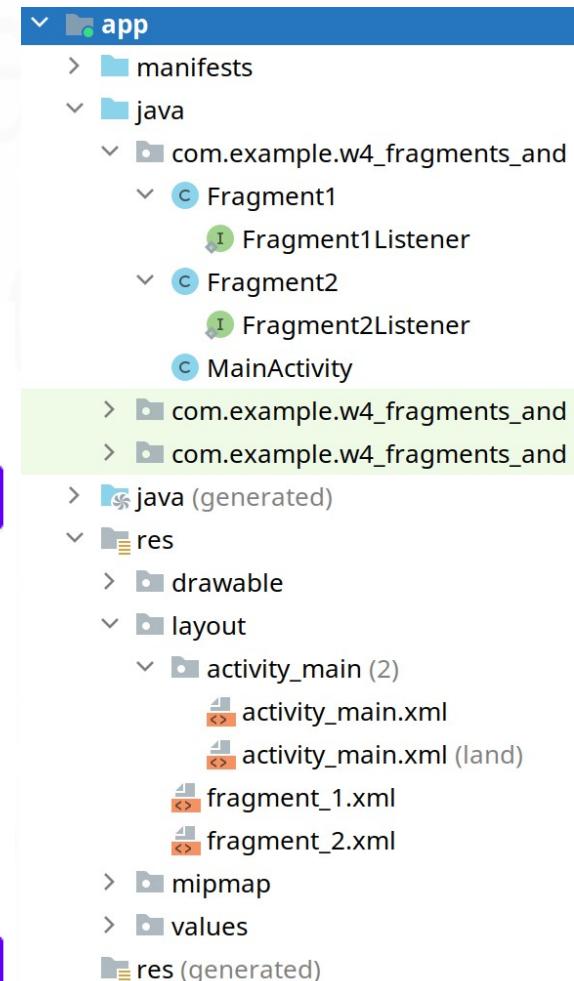
Fragment 1

SEND TO ACTIVITY

Nothing so far...

Nothing so far...

FORWARD TO FRAGMENTS



# Fragmenty – aktywność

```
//bardzo ważne - zaimplementowane interfejsy
public class MainActivity extends AppCompatActivity implements
Fragment1.Fragment1Listener,
    Fragment2.Fragment2Listener {

    //standardowa metoda onCreate()
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //obsługa zdarzeń
        Button forwardButton =
            (Button) findViewById(R.id.forward_button);
        forwardButton.setOnClickListener(v -> forwardMessages());
    }

    //implementacja interfejsów
    @Override
    public void messageFromFragment1(String message) {
        TextView textFrom1 =
            (TextView) findViewById(R.id.text_from_1_text_view);
        textFrom1.setText(message);
    }
}
```

# Fragmenty – aktywność

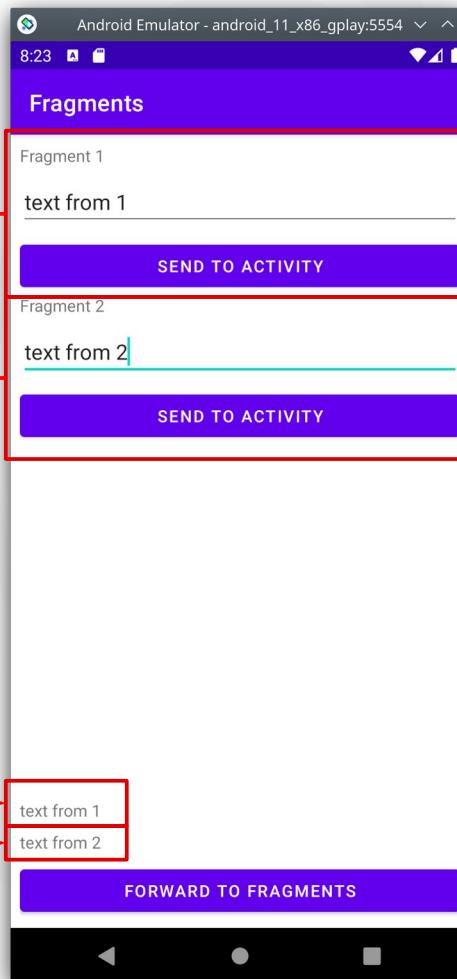
```
@Override  
public void messageFromFragment2(String message) {  
    TextView textFrom2 =  
        (TextView) findViewById(R.id.text_from_2_text_view);  
    textFrom2.setText(message);  
  
}  
  
// metody pomocnicze sprawdzające czy istnieje fragment  
private Fragment1 getFragment1() {  
    Fragment1 fragment1 = (Fragment1) getSupportFragmentManager()  
        .findFragmentById(R.id.fragment1);  
// fragment może istnieć ale nie być widoczny  
    if (fragment1 != null && fragment1.isAdded())  
        return fragment1;  
    return null;  
}  
  
private Fragment2 getFragment2() {  
    // analogicznie jak dla fragmentu 1...  
}
```

# Fragmenty – aktywność

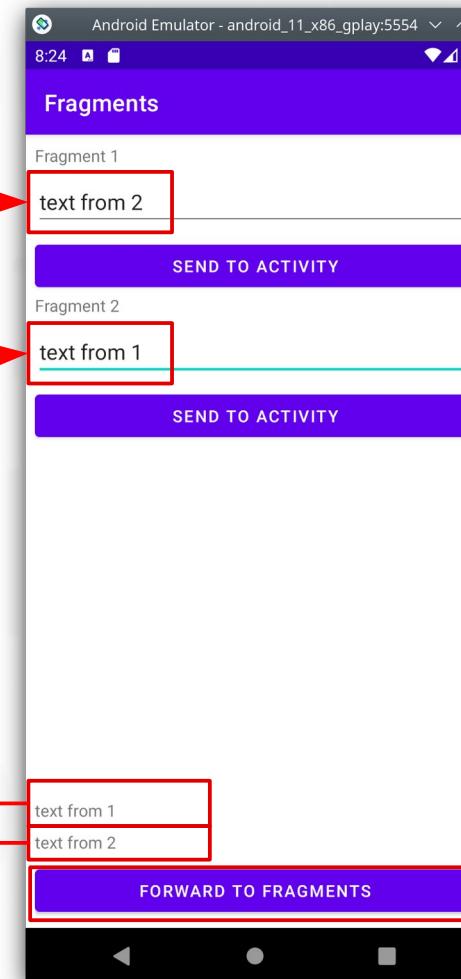
```
//wysyłanie danych do fragmentów (komunikacja między fragmentami)
private void forwardMessages() {
    String textFrom1 =
        ((TextView) findViewById(R.id.text_from_1_text_view))
            .getText().toString();
    String textFrom2 =
        ((TextView) findViewById(R.id.text_from_2_text_view))
            .getText().toString();
    Fragment1 fragment1 = getFragment1();
    if (fragment1 != null)
        fragment1.setText(textFrom2);
    Fragment2 fragment2 = getFragment2();
    if (fragment2 != null)
        fragment2.setText(textFrom1);
}
```

# Fragmenty

1

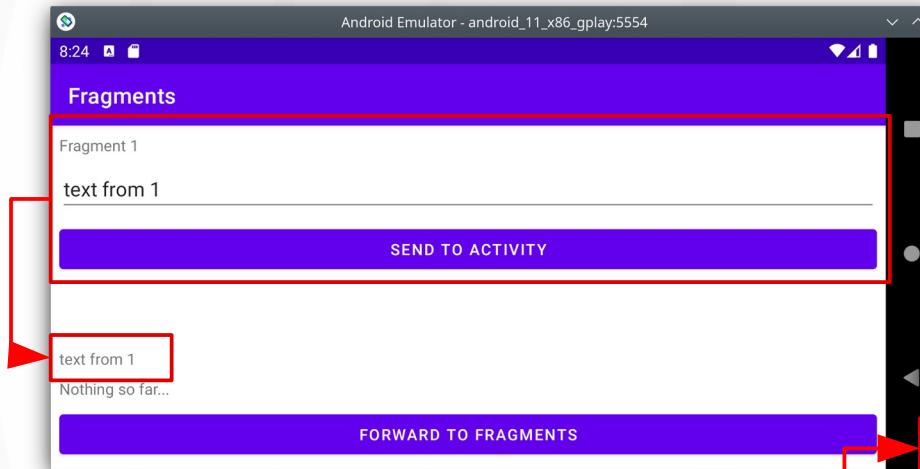


2

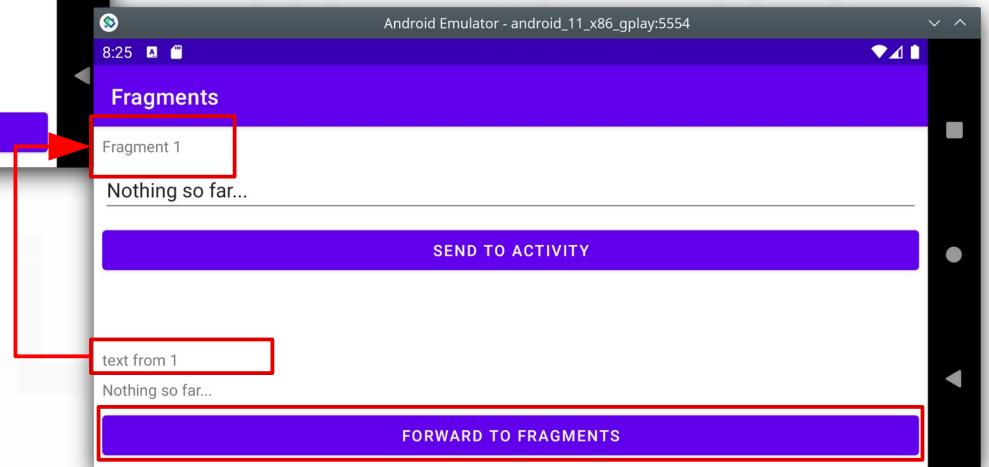


# Fragmenty

1



2



# Okna dialogowe

- Istnieją różne dwa sposoby tworzenia okien dialogowych
- Zgodnie z zalecanyim sposobem tworzenia okien dialogowych należy
  - utworzyć klasę pochodną DialogFragment,
  - utworzyć układ w pliku XML opisujący zawartość okna dialogowego
- Klasa okna dialogowego zazwyczaj definiuje standardowe elementy fragmentu takie jak: interfejs, onAttach(), onDetach() a także charakterystyczną dla okien dialogowych: onCreateDialog()
- Okna stworzone tą metodą są odtwarzane po obróceniu urządzenia

# Okna dialogowe – layout, activity\_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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=".MainActivity">

    <TextView
        android:id="@+id/dialog_value_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Value from dialog"
        android:textSize="30sp"
        app:layout_constraintBottom_toBottomOf=
            "parent"
        app:layout_constraintEnd_toEndOf=
            "parent"
        app:layout_constraintStart_toStartOf=
            "parent"
        app:layout_constraintTop_toTopOf=
            "parent" />
```

Value from dialog

SHOW DIALOG

# Okna dialogowe – layout, activity\_main.xml

Value from dialog

SHOW DIALOG

```
<Button  
    android:id="@+id/show_dialog_button"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below=  
        "@+id/dialog_value_text_view"  
    android:layout_alignLeft=  
        "@+id/dialog_value_text_view"  
    android:layout_marginTop="32dp"  
    android:text="Show dialog"  
    app:layout_constraintEnd_toEndOf=  
        "parent"  
    app:layout_constraintStart_toStartOf=  
        "parent"  
    app:layout_constraintTop_toBottomOf=  
        "@+id/dialog_value_text_view" />
```

```
</androidx.constraintlayout  
.widget.ConstraintLayout>
```

# Okna dialogowe – layout, value\_dialog.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    xmlns:tools=
        "http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="8dp"
    tools:context=".MainActivity">

    <EditText
        android:id=
            "@+id/value_dialog_edit_text"
        android:inputType="number"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <requestFocus />
    </EditText>
</LinearLayout>
```

# Okna dialogowe – aktywność

```
public class MainActivity extends AppCompatActivity implements  
ValueDialogFragment.ValueDialogFragmentListener {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        Button showDialogButton = findViewById(R.id.show_dialog_button);  
        showDialogButton.setOnClickListener(v -> showDialog());  
    }  
    //etykieta dialogu (getSupportFragmentManager().getFragmentByTag())  
  
    public final static String VALUE_DIALOG_TAG =  
        "com.example.w4_dialogs_and.value_dialog";  
  
    private void showDialog() {  
        DialogFragment valueDialogFragment = new ValueDialogFragment();  
        valueDialogFragment.show(getSupportFragmentManager(),  
            VALUE_DIALOG_TAG);  
    }  
    //reakcja na kliknięcie OK w dialogu / implementacja interfejsu  
    @Override  
    public void valueEnteredInDialog(int wartosc) {  
        TextView dialogValueTextView =  
            findViewById(R.id.dialog_value_text_view);  
        dialogValueTextView.setText(Integer.toString(wartosc));  
    }  
}
```

# Okna dialogowe – ValueDialogFragment.java

```
public class ValueDialogFragment extends DialogFragment {  
  
    private ValueDialogFragmentListener mListener;  
  
    public interface ValueDialogFragmentListener {  
        // wysyłanie wprowadzonej wartości do aktywności  
        public void valueEnteredInDialog(int wartosc);  
    }  
  
    public ValueDialogFragment() {}  
  
    // podłączenie do aktywności  
    @Override  
    public void onAttach(Context context) {  
        super.onAttach(context);  
        try {  
            mListener = (ValueDialogFragmentListener) context;  
        } catch (ClassCastException e) {  
            throw new ClassCastException(context.toString()  
                + " must implement ValueDialogFragmentListener");  
    } }  
}
```

# Okna dialogowe – ValueDialogFragment.java

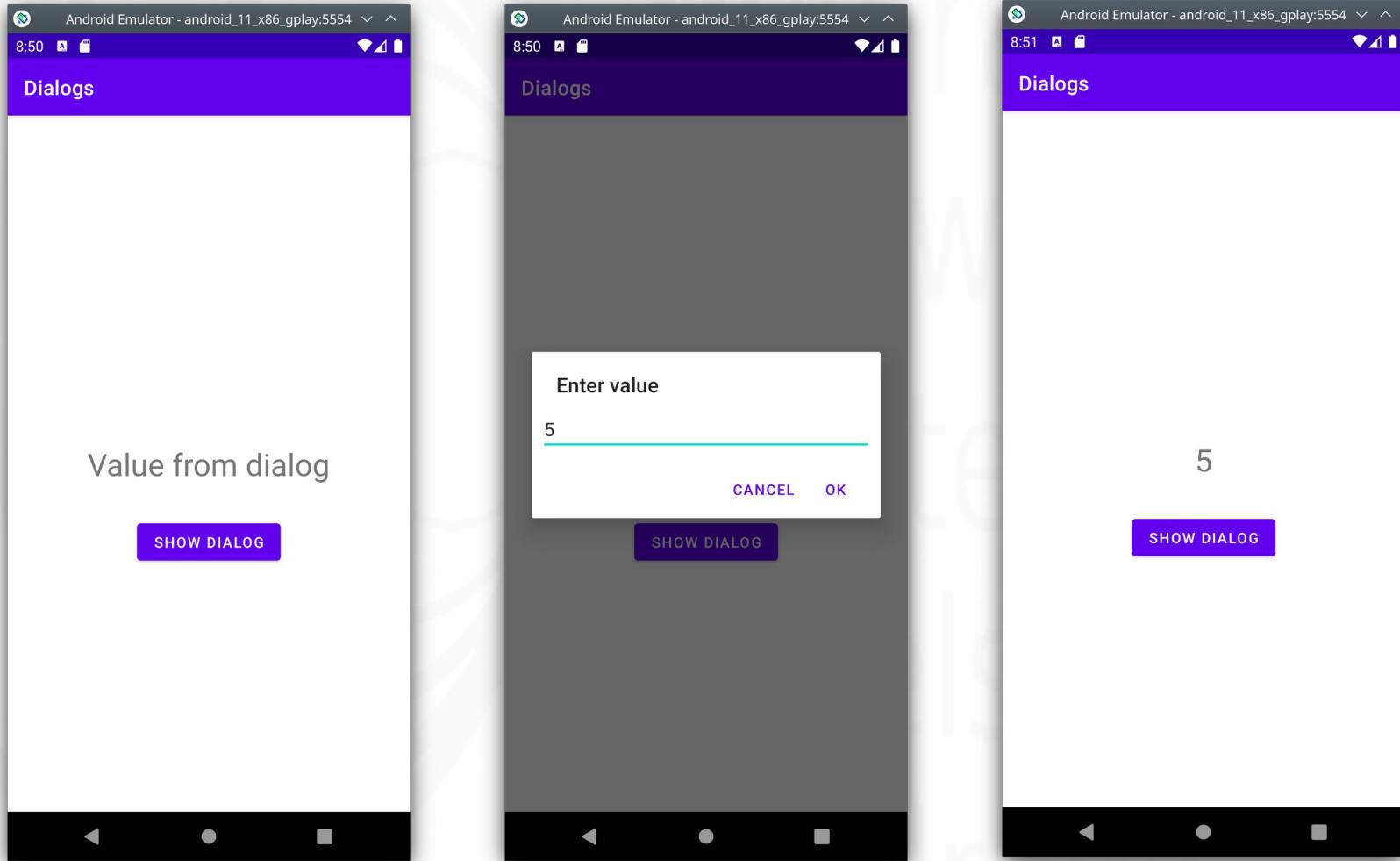
```
// odłączenie od aktywności
@Override
public void onDetach() {
    super.onDetach();
    mListener = null;
}

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    AlertDialog.Builder builder =
        new AlertDialog.Builder(getActivity());
    LayoutInflater inflater =
        getActivity().getLayoutInflater();
    builder
        // utworzenie układu dialogu
        .setView(inflater.inflate(
            R.layout.value_dialog, null))
        // ustawienie tytułu
        .setTitle("Enter value")
```

# Okna dialogowe – ValueDialogFragment.java

```
// obsługa przycisku zatwierdzającego
.setPositiveButton("OK", (dialog, id) -> {
    // konwersja wartości z pola do int i wysłanie do
    // aktywności wartości zwrócić uwagę na getDialog()
    // przed findViewById()
    EditText valueEditText = getDialog()
        .findViewById(R.id.value_dialog_edit_text);
    String wartoscStr =
        valueEditText.getText().toString();
    int value = 0;
    try {
        value = Integer.parseInt(wartoscStr);
    } catch (NumberFormatException ignored) {}
    mListener.valueEnteredInDialog(value);
})
// obsługa przycisku anulującego
.setNegativeButton("Cancel", (dialog, id) -> {
    // anulowanie dialogu
    ValueDialogFragment.this
        .getDialog().cancel();
});
// utworzenie układu dialogu
return builder.create();
}}
```

# Okna dialogowe – ValueDialogFragment.java





# Programowanie Aplikacji Mobilnych na Platformę Android

## Wykład 5 – Sposoby trwałego przechowywania danych na potrzeby aplikacji mobilnych.

Jakub Smołka



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny

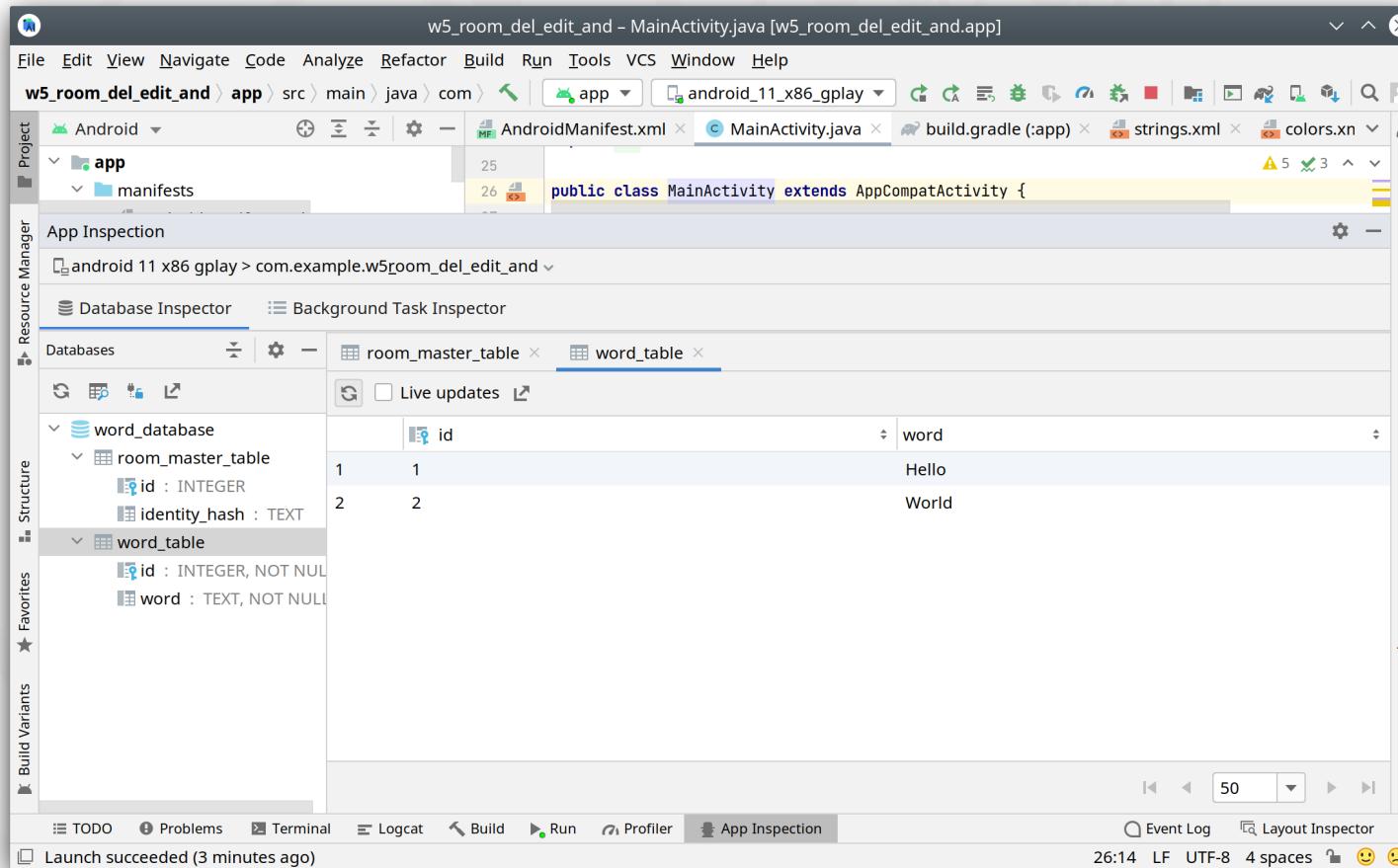


# Baza danych SQLite

- SQLite to biblioteka napisana w języku C. Implementuje mały, szybki, zwarty, niezawodny silnik SQL o dużych możliwościach
- Jest powszechnie używany m. in. w systemach Android i iOS, oraz przeglądarkach Firefox, Chrome, Opera, Safari
- Aktywnie używanych jest ponad 1 bilion ( $10^{12}$ ) baz SQLite
- Oferuje zbiór właściwości ACID (niepodzielność, spójność, izolacja, trwałość)
- W systemie Android dostępne jest polecenie sqlite3 oraz biblioteka, którą można wykorzystywać we własnych aplikacjach

# Badanie zawartości bazy danych

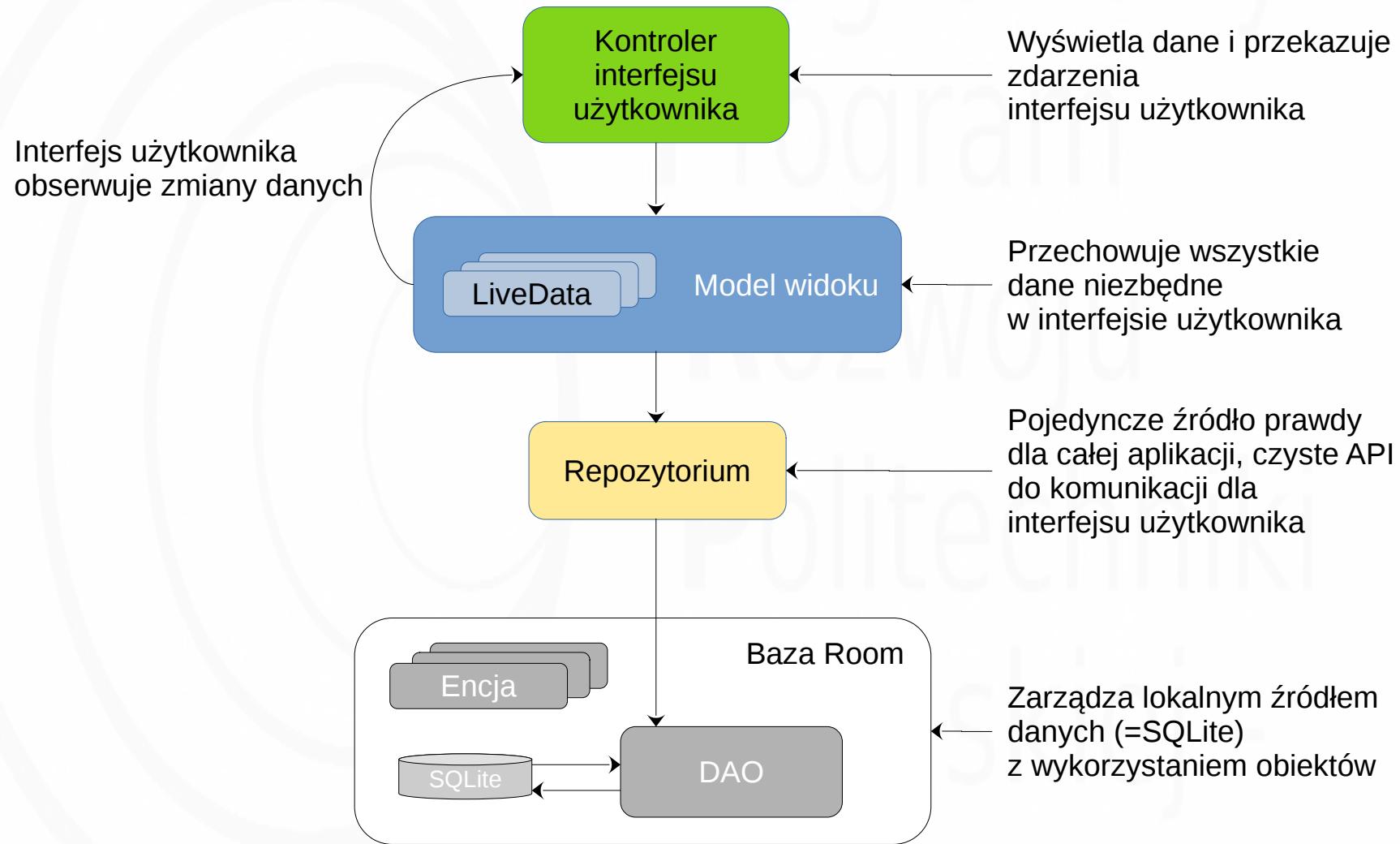
- Na dolnym pasku wybrać przycisk *App Inspection*
- Wybrać aplikację z bazą danych



# Biblioteka Room

- Biblioteka *Room* (*Room Persistence Library*) zapewnia warstwę abstrakcji nad bazą *SQLite*
- Może zostać wykorzystana do stworzenia lokalnego bufora danych pobranych z Internetu
- Zapewnia mapowanie obiektowo relacyjne. Rekordy pobrane z bazy są przekształcane w obiekty a obiekty wysyłane do bazy automatycznie przekształcane są na postać zrozumiałą dla bazy *SQLite*
- Baza i tabele są tworzone automatycznie (na podstawie stworzonych przez programistę klas)
- Podstawowe zapytania również mogą być generowane automatycznie

# Biblioteka Room – struktura aplikacji



źródło: <https://developer.android.com/codelabs/android-training-livedata-viewmodel/img/fd28069527c8d615.png>

# Biblioteka Room – pojęcia

- *Entity* – Reprezentuje rekordy w tabeli
  - na podstawie tej klasy biblioteka Room tworzy tabelę w bazie danych
  - właściwości kolumn określane są za pomocą adnotacji
- *DAO* (data access object) – interfejs/klasa abstrakcyjna
  - mapuje zapytania SQL na metody umożliwiające odczytywanie danych z bazy lub ich zapisywanie do bazy
  - klasa implementująca metody opisane w interfejsie/klasie abstrakcyjnej jest tworzona automatycznie przez bibliotekę Room

# Biblioteka Room – pojęcia

- *RoomDatabase* – klasa abstrakcyjna
  - implementuje wzorzec singleton
  - zapewnia dostęp do obiektu DAO
  - wykonuje zadania pomocnicze takie jak tworzenie bazy danych (przy pierwszym uruchomieniu) czy wykonująca migrację do nowej wersji
- *Repozytorium* – źródło danych dla aplikacji
  - może pobierać dane z wielu źródeł
  - zapewnia spójny interfejs dla całej aplikacji
  - jednym ze źródeł jest typowo baza danych SQLite (korzystamy z obiektu DAO, nie z bazy bezpośrednio)

# Biblioteka Room – pojęcia

- *View model* – model widoku – inaczej klasa przechowująca dane prezentowane w widoku czyli np. na liście wyświetlanej użytkownikowi
  - zaleta – czas życia tego obiektu jest dłuższy niż aktywności (model widoku przetrwa zniszczenie i odtworzenie aktywności przy obróceniu telefonu)
- *Live data* – klasa pełniąca rolę pojemnika na dane
  - realizuje wzorzec *obserwatora* – aktywność zgłasza (rejestruje się), że jest zainteresowana zmianami danych i będzie powiadamiana, jeżeli dane w pojemniku się zmienią
  - obiekty live data uwzględniają cykl życia aktywności (i innych elementów np. fragmentów)

# Biblioteka Room – zależności, build.gradle (moduł)

```
//...  
dependencies {  
    //...  
  
    // Selection Tracker - zaznaczanie elementów listy  
    implementation "androidx.recyclerview:recyclerview-selection:1.1.0"  
  
    // składniki biblioteki Room  
    implementation "androidx.room:room-runtime:2.4.0"  
    annotationProcessor "androidx.room:room-compiler:2.4.0"  
    androidTestImplementation "androidx.room:room-testing:2.4.0"  
}
```

# Biblioteka Room – layouty

Item 0

Item 1

Item 2

Item 3

Item 4

Item 5

Item 6

Item 7

Item 8

- Układ głównej aktywności jest bardzo prosty zawiera tylko listę
  - `activity_main.xml`
  - identyfikator: `recyclerview`
- Układ wiersza listy również jest bardzo prosty zawiera tylko jeden istotny element
  - `recyclerview_item.xml`
  - identyfikator: `wordTextView`

# Biblioteka Room – opis tabeli (entity)

- Opisuje tabelę i rekordy przechowywane w tabeli
- Na jej podstawie Room tworzy tabelę

```
@Entity(tableName = "word_table")
public class Word {
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id")
    private long mId;

    @NonNull
    @ColumnInfo(name = "word")
    private String mWord;

    public Word(@NonNull String word) {
        mWord = word;
    }
    //niektóre gettery i settery są wymagane przez Room
    public long getId() { return mId; }
    @NonNull
    public String getWord() { return mWord; }
    public void setId(long mId) { this.mId = mId; }
}
```

# Biblioteka Room – DAO

- Opisuje operacje wykonywane na rekordach
- Może być interfejsem lub klasą abstrakcyjną
- Implementacja jest tworzona automatycznie przez Room

```
@Dao
public interface WordDao {
    //aby zezwolić na wielokrotne dodanie identycznego rekordu
    //zmienić strategię rozwiązywania konfliktów -> IGNORE
    @Insert(onConflict = OnConflictStrategy.ABORT)
    void insert(Word word);

    @Query("DELETE FROM word_table")
    void deleteAll();
    //zapytania Room są wykonywane w osobnym wątku
    //LiveData powiadamia obserwatora w głównym wątku
    @Query("SELECT * FROM word_table ORDER BY word ASC")
    LiveData<List<Word>> getAlphabetizedWords();

    @Delete
    void deleteWord(Word word);
}
```

# Biblioteka Room – RoomDatabase

- Klasa RoomDatabase pełni rolę pomocniczą (otwieranie, migracja danych ...)
- Abstrakcyjna - implementacja niektórych elementów jest tworzona automatycznie przez Room

```
//musi być abstrakcyjna
//określamy listę klas reprezentujących tabele, wersję bazy,
//klasa może obsługiwać migrację do nowej wersji bazy
@Database(entities = {Word.class}, version = 1, exportSchema = false)
public abstract class WordRoomDatabase extends RoomDatabase {
    public abstract WordDao wordDao();

    //singleton
    private static volatile WordRoomDatabase INSTANCE;
    private static final int NUMBER_OF_THREADS = 4;

    //wykonanie w tle
    static final ExecutorService databaseWriteExecutor =
        Executors.newFixedThreadPool(NUMBER_OF_THREADS);
```

# Biblioteka Room – RoomDatabase

```
static WordRoomDatabase getDatabase(final Context context) {  
    if (INSTANCE == null) {  
        synchronized (WordRoomDatabase.class) {  
            if (INSTANCE == null) {  
                INSTANCE = Room.databaseBuilder(  
                    context.getApplicationContext(),  
                    WordRoomDatabase.class, "word_database")  
                    .addCallback(sRoomDatabaseCallback)  
                    .fallbackToDestructiveMigration()  
                    .build();  
            }  
        }  
    }  
    return INSTANCE;  
}  
  
private static RoomDatabase.Callback sRoomDatabaseCallback =  
    new RoomDatabase.Callback() {  
        //jest również metoda onOpen()  
        @Override  
        public void onCreate(@NonNull SupportSQLiteDatabase db) {  
            super.onCreate(db);  
            databaseWriteExecutor.execute(() -> {  
                WordDao dao = INSTANCE.wordDao();  
                Word[] words = {new Word("Hello"), new Word("World")};  
                for (Word w : words)  
                    dao.insert(w);  
            });  
        };  
    };
```

# Biblioteka Room – repozytorium

- Repozytorium zarządza zapytaniami. Może korzystać z wielu źródeł danych
- W najbardziej typowym przypadku repozytorium implementuje logikę decydującą czy pobrać dane z lokalnej bazy czy z sieci
- W przykładzie korzystamy tylko z lokalnej bazy SQLite

```
public class WordRepository {  
    private WordDao mWordDao;  
    private LiveData<List<Word>> mAllWords;  
  
    WordRepository(Application application) {  
        WordRoomDatabase wordRoomDatabase =  
            WordRoomDatabase.getDatabase(application);  
        //repozytorium korzysta z DAO a nie bezpośrednio z bazy  
        mWordDao = wordRoomDatabase.wordDao();  
        mAllWords = mWordDao.getAlphabetizedWords();  
    }  
  
    LiveData<List<Word>> getAllWords() { return mAllWords; }  
}
```

# Biblioteka Room – repozytorium

```
//operacje muszą być wykonywane w osobnym wątku

void insert(Word word) {
    //Runnable -> Lambda
    WordRoomDatabase.databaseWriteExecutor.execute(() -> {
        mWordDao.insert(word);
    });
}

void deleteAll() {
    WordRoomDatabase.databaseWriteExecutor.execute(() -> {
        mWordDao.deleteAll();
    });
}

void deleteWord(Word word)
{
    WordRoomDatabase.databaseWriteExecutor.execute(() -> {
        mWordDao.deleteWord(word);
    });
}
```

# Biblioteka Room – adapter dla RecyclerView

```
public class WordListAdapter extends  
RecyclerView.Adapter<WordListAdapter.WordViewHolder> {  
  
    private LayoutInflator mLayoutInflater;  
    private List<Word> mWordList;  
  
    public WordListAdapter(Context context) {  
        mLayoutInflater=LayoutInflater.from(context);  
        this.mWordList = null;  
    }  
  
    @NonNull  
    @Override  
    public WordViewHolder onCreateViewHolder(@NonNull ViewGroup  
parent, int viewType) {  
        View rootView= mLayoutInflater  
            .inflate(R.layout.recyclerview_item,parent,false);  
        WordViewHolder wordViewHolder=new WordViewHolder(rootView);  
        return wordViewHolder;  
    }  
}
```

# Biblioteka Room – adapter dla RecyclerView

```
@Override  
public void onBindViewHolder(@NonNull WordViewHolder holder,  
    int position) {  
    holder.bindToWordViewHolder(position);  
}  
  
@Override  
public int getItemCount() {  
    if (mWordList!=null)  
        return mWordList.size();  
    return 0;  
}  
  
public class WordViewHolder extends RecyclerView.ViewHolder {  
  
    TextView wordTextView;  
  
    public WordViewHolder(@NonNull View itemView) {  
        super(itemView);  
        wordTextView=itemView.findViewById(R.id.wordTextView);  
    }  
}
```

# Biblioteka Room – adapter dla RecyclerView

```
public void bindToWordViewHolder(int position)
{
    wordTextView.setText(mWordList.get(position).getWord());
}
}

public void setWordList(List<Word> wordList) {
    this.mWordList = wordList;
    notifyDataSetChanged();
}
```

# Biblioteka Room – model widoku

```
//AndroidViewModel przechowuje kontekst aplikacji  
//ViewModel nie ma kontekstu  
public class WordViewModel extends AndroidViewModel {  
    private final WordRepository mRepository;  
    private final LiveData<List<Word>> mAllWords;  
  
    public WordViewModel(@NotNull Application application) {  
        super(application);  
        mRepository = new WordRepository(application);  
        mAllWords = mRepository.getAllWords();  
    }  
  
    LiveData<List<Word>> getAllWords() { return mAllWords; }  
  
    public void insert(Word word) { mRepository.insert(word); }  
  
    public void deleteAll() { mRepository.deleteAll(); }  
  
    public void deleteWord(Word word)  
    {  
        mRepository.deleteWord(word);  
    }  
}
```

# Biblioteka Room – główna aktywność

```
public class MainActivity extends AppCompatActivity {

    private WordViewModel mWordViewModel;
    private WordListAdapter mAdapter;

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

        RecyclerView recyclerView = findViewById(R.id.recyclerview);
        mAdapter = new WordListAdapter(this);
        recyclerView.setAdapter(mAdapter);

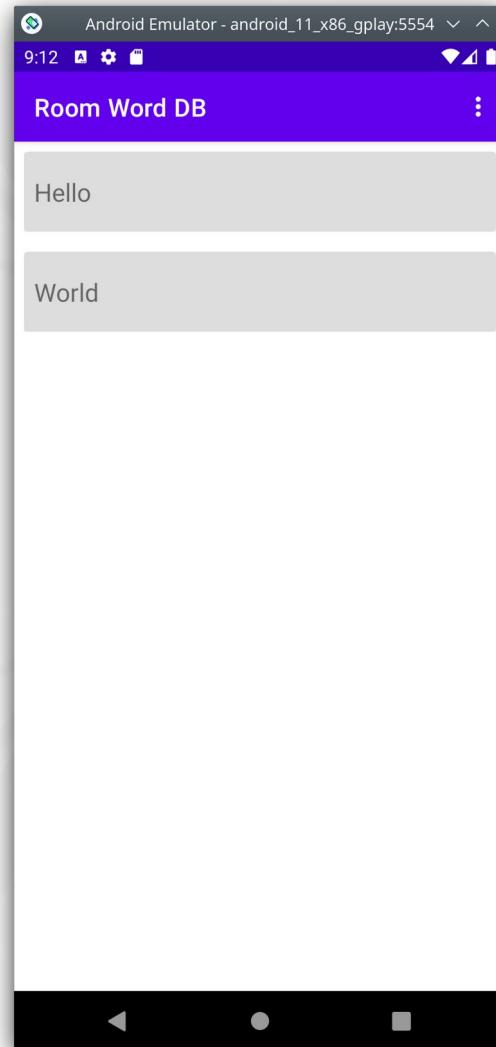
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        mWordViewModel = new ViewModelProvider(this)
            .get(WordViewModel.class);

        //Observer::onChanged -> Lambda
        mWordViewModel.getAllWords().observe(this, words -> {
            mAdapter.setWordList(words);
        });
    }
}
```

# Biblioteka Room – główna aktywność

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    getMenuInflater().inflate(R.menu.main,menu);  
    return super.onCreateOptionsMenu(menu);  
}  
  
@Override  
public boolean onOptionsItemSelected(@NonNull MenuItem item) {  
    int id = item.getItemId();  
  
    if (id == R.id.clear_data)  
    {  
        Toast.makeText(this,"Clearing the data...",  
            Toast.LENGTH_SHORT).show();  
        mWordViewModel.deleteAll();  
        return true;  
    }  
    return super.onOptionsItemSelected(item);  
}
```

# Biblioteka Room – wyświetlanie rekordów



# Biblioteka Room – dodawanie rekordów, modyfikacja layoutu



- Układ głównej aktywności został zmodyfikowany
  - activity\_main.xml
  - recyclerview
  - fabMain
  - fabDelete
- Przycisk fabDelete (domyślnie ukryty) będzie wykorzystywany w kolejnym przykładzie

# Biblioteka Room – dodawanie rekordów, modyfikacja layoutu

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    <!-- ... -->
    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fabMain"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:src="@drawable/ic_baseline_add_24"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="16dp" />

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fabDelete"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:src="@drawable/ic_baseline_delete_24"
        android:visibility="gone"
        app:fabSize="mini"
        app:layout_constraintBottom_toTopOf="@+id/fabMain"
        app:layout_constraintEnd_toEndOf="@+id/fabMain"
        app:layout_constraintStart_toStartOf="@+id/fabMain" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

# Biblioteka Room – dodawanie rekordów, modyfikacje głównej aktywności

```
public class MainActivity extends AppCompatActivity {  
    //...  
    private FloatingActionButton mMainFab;  
    //...  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        //...  
        mActivityResultLauncher = registerForActivityResult(  
            new ActivityResultContracts.StartActivityForResult(),  
            result -> {  
                if (result.getResultCode() == RESULT_OK) {  
                    Word word = new Word(result.getData()  
                        .getStringExtra(  
                            NewWordActivity.EXTRA_REPLY));  
                    mWordViewModel.insert(word);  
                }  
            });  
        mMainFab = findViewById(R.id.fabMain);  
        mMainFab.setOnClickListener(view -> mainFabClicked());  
        //...  
    }  
}
```

# Biblioteka Room – dodawanie rekordów, modyfikacje głównej aktywności

```
//...
ActivityResultLauncher<Intent> mActivityResultLauncher;

private void mainFabClicked() {
    Intent intent = new Intent(
        MainActivity.this, NewWordActivity.class);
    mActivityResultLauncher.launch(intent);
}
//...
```

# Biblioteka Room – layout aktywności dodawania nowego słowa

Enter new word...

SAVE

- Układ aktywności nowego słowa jest prosty zawiera dwa istotne elementy
  - `activity_new_word.xml`
    - `edit_word`
    - `button_save`

# Biblioteka Room – aktywność dodawania nowego słowa

```
public class NewWordActivity extends AppCompatActivity {

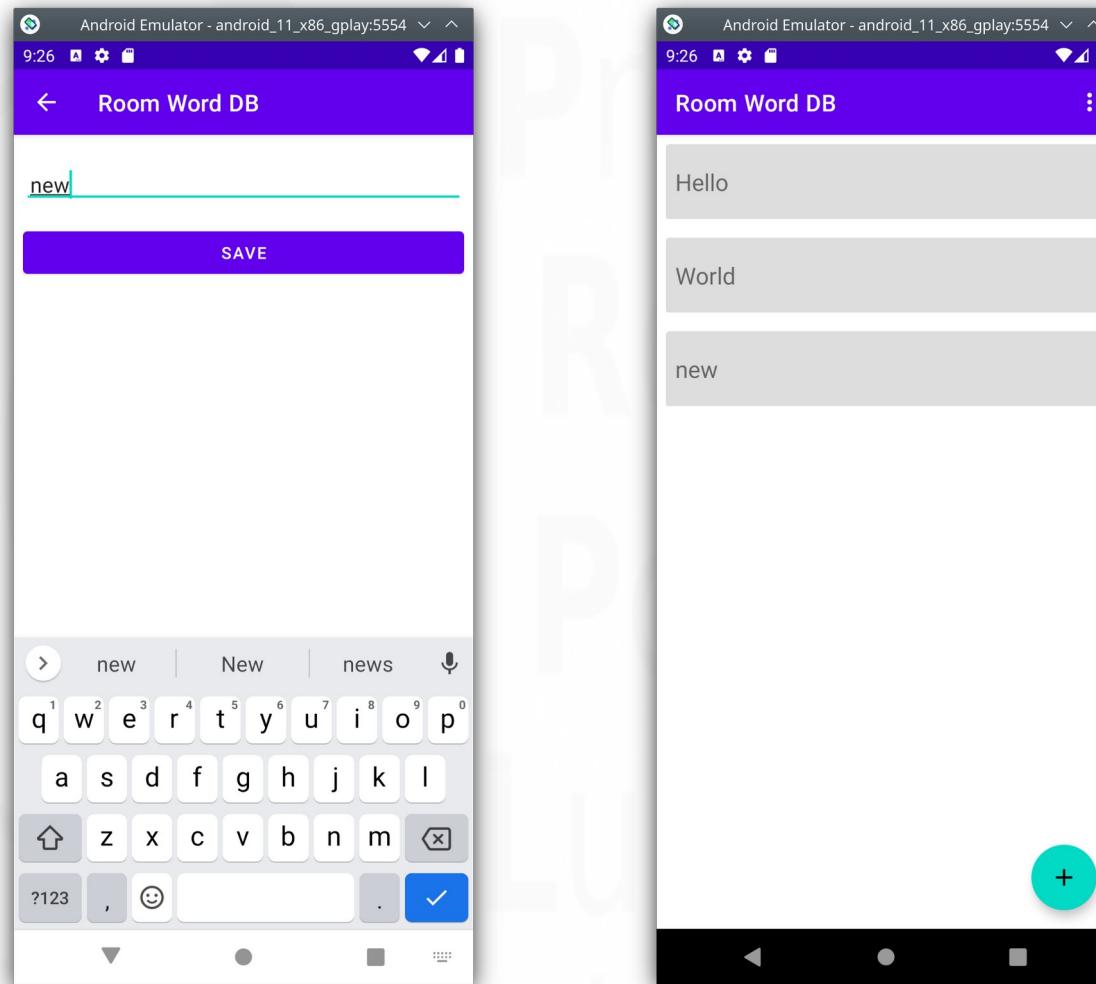
    public static final String EXTRA_REPLY =
        "com.example.android.wordlistsql.REPLY";

    private EditText mEditWordText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_new_word);
        mEditWordText = findViewById(R.id.edit_word);

        final Button button = findViewById(R.id.button_save);
        button.setOnClickListener(view -> {
            if (TextUtils.isEmpty(mEditWordText.getText().toString())) {
                mEditWordText.setError(getString(R.string.error_word));
                return;
            }
            Intent replyIntent = new Intent();
            String word = mEditWordText.getText().toString();
            replyIntent.putExtra(EXTRA_REPLY, word);
            setResult(RESULT_OK, replyIntent);
            finish();
        });
    }
}
```

# Biblioteka Room – dodawanie nowego słowa



# Biblioteka Room – zaznaczalne elementy

- Layout `recyclerview_item.xml` musi być zmodyfikowany aby reagował za zaznaczenie zmianą koloru

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="8dp">

    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="64dp"
        app:cardCornerRadius="4dp">

        <TextView
            android:id="@+id/wordTextView"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:paddingStart="8dp"
            android:gravity="center_vertical"
            android:textSize="20sp"
            android:background="@drawable/list_background" />
    </androidx.cardview.widget.CardView>
</LinearLayout>
```

# Biblioteka Room – zaznaczalne elementy

- Element listy nie określa bezpośrednio jak ma się zmieniać kolor elementu
- Odpowiedzialna jest za to *lista stanów/selektor* ustawiona jako tło
- Stan aktywny elementu = zaznaczony
- Lista stanów: list\_background.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@color/purple_500"
          android:state_activated="true"></item>

    <item android:drawable="@color/custom_light_gray"></item>
</selector>
```

# Biblioteka Room – zaznaczanie i kasowanie elementów, modyfikacje głównej aktywności

```
public class MainActivity extends AppCompatActivity {

    //...
    //typ == klucz
    private SelectionTracker<Long> mSelectionTracker;
    private FloatingActionButton mDeleteFab;
    private boolean mIsMainFabAdd = true;
    private WordItemKeyProvider mWordItemKeyProvider;
    //...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //...
        //bardzo ważne trzeba utworzyć i ustawić adapter przed
        //obiektem SelectrionTracker
        //key provider - umożliwia ustalenie klucza wybranego elementu
        mWordItemKeyProvider = new WordItemKeyProvider();
        mAdapter = new WordListAdapter(this,mWordItemKeyProvider);
        recyclerView.setAdapter(mAdapter);
    }
}
```

# Biblioteka Room – zaznaczanie i kasowanie elementów, modyfikacje głównej aktywności

```
mSelectionTracker = new SelectionTracker.Builder<>(
    "word-selection",
    recyclerView,
    mWordItemKeyProvider,
    //odczytuje szczegóły wybranego elementu
    new WordItemDetailsLookup(recyclerView),
    //magazyn na klucze
    StorageStrategy.createLongStorage()).build();
mAdapter.setSelectionTracker(mSelectionTracker);
//...
//obsługa przycisku FAB kasowania zaznaczonych elementów
mDeleteFab = findViewById(R.id.fabDelete);
mDeleteFab.setOnClickListener(view -> deleteSelection());

//selection tracker będzie informował o zmianach zaznaczenia
mSelectionTracker.addObserver(
    new SelectionTracker.SelectionObserver<Long>() {
        @Override
        public void onSelectionChanged() {
            //wyświetlamy/chowamy przycisk kasowania
            updateFabs();
            super.onSelectionChanged();
        }
    }
)
```

# Biblioteka Room – zaznaczanie i kasowanie elementów, modyfikacje głównej aktywności

```
@Override  
public void onSelectionRestored() {  
    //wyświetlamy/chowamy przycisk kasowania  
    updateFabs();  
    super.onSelectionRestored();  
}  
@Override  
public void onItemStateChanged(@NonNull Long key,  
    boolean selected) {  
    super.onItemStateChanged(key, selected);  
});}  
//...  
private void mainFabClicked() {  
    //główny przycisk może być „+” (dodaj) lub „x” (anuluj)  
    if (mIsMainFabAdd) {  
        Intent intent = new Intent(  
            MainActivity.this, NewWordActivity.class);  
        mActivityResultLauncher.launch(intent);  
    } else {  
        mSelectionTracker.clearSelection();  
    }  
}
```

# Biblioteka Room – zaznaczanie i kasowanie elementów, modyfikacje głównej aktywności

```
private void deleteSelection() {
    Selection<Long> selection=mSelectionTracker.getSelection();
    int wordPosition=-1;
    List<Word> wordList=mWordViewModel.getAllWords().getValue();
    //przeglądamy identyfikatory z zaznaczenia i kasujemy elementy
    for (long wordId:selection) {
        wordPosition=mWordItemKeyProvider.getPosition(wordId);
        mWordViewModel.deleteWord(wordList.get(wordPosition));
    }
}

//w zależności od sytuacji główny przycisk jest „+” lub „x”
//dodatkowo pokazujemy lub ukrywamy przycisk kasowania
private void updateFabs() {
    if (mSelectionTracker.hasSelection())
    {
        mDeleteFab.setVisibility(View.VISIBLE);
        mMainFab.setImageDrawable(
            getDrawable(R.drawable.ic_baseline_cancel_24));
        mIsMainFabAdd=false;
    }
}
```

# Biblioteka Room – zaznaczanie i kasowanie elementów, modyfikacje głównej aktywności

```
        else
    {
        mDeleteFab.setVisibility(View.GONE);
        mMainFab.setImageDrawable(
            getDrawable(R.drawable.ic_baseline_add_24));
        mIsMainFabAdd=true;
    }
}
@Override
protected void onSaveInstanceState(@NonNull Bundle outState) {
    super.onSaveInstanceState(outState);
    //stan zaznaczenia trzeba zachować
    mSelectionTracker.onSaveInstanceState(outState);
}
@Override
protected void onRestoreInstanceState(
    @NonNull Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    //stan zaznaczenia trzeba przywrócić
    mSelectionTracker.onRestoreInstanceState(savedInstanceState);
}
//...
}
```

# Biblioteka Room – zaznaczanie i kasowanie elementów, szczegółowe dane wybranego elementu

```
//prosta klasa przechowująca szczegóły elementu list
//pozycję i klucz (identyfikator) typu Long
public class WordItemDetails extends
ItemDetailsLookup.ItemDetails<Long> {
    int position;
    long id;
    @Override
    public int getPosition() { return position; }

    @Nullable
    @Override
    public Long getSelectionKey() { return id; }

    @Override
    public boolean inSelectionHotspot(@NonNull MotionEvent e) {
        return false;
    }
    @Override
    public boolean inDragRegion(@NonNull MotionEvent e) {
        return true;
    }
}
```

# Biblioteka Room – zaznaczanie i kasowanie elementów, ustalanie wybranego elementu

```
public class WordItemDetailsLookup extends ItemDetailsLookup<Long> {  
    private RecyclerView mRecyclerView;  
    public WordItemDetailsLookup(RecyclerView mRecyclerView) {  
        this.mRecyclerView = mRecyclerView;  
    }  
    @Nullable  
    @Override  
    public ItemDetails<Long> getItemDetails(@NonNull MotionEvent e) {  
        //ustala szczegóły zaznaczonego elementu (na podstawie x i y)  
        View view=  
            mRecyclerView.findChildViewUnder(e.getX(),e.getY());  
        if (view !=null)  
        {  
            RecyclerView.ViewHolder viewHolder=  
                mRecyclerView.getChildViewHolder(view);  
            if (viewHolder instanceof WordListAdapter.WordViewHolder)  
                return ((WordListAdapter.WordViewHolder) viewHolder)  
                    .getWordItemDetails();  
        }  
        return null;  
    }  
}
```

# Biblioteka Room – zaznaczanie i kasowanie elementów, mapowanie klucz-pozycja elementu

```
//Klasa zapewnia mapowanie pomiędzy kluczami Long a pozycją na liście
//na podstawie pozycji ustala klucz a na podstawie klucza ustala pozycję
public class WordItemKeyProvider extends ItemKeyProvider<Long> {
    private Map<Long, Integer> mKeyToPosition;
    private List<Word> mWordList;
    public WordItemKeyProvider() {
        super(SCOPE_MAPPED);
        mWordList=null;
    }
    public void setWords(List<Word> wordList)
    {
        this.mWordList = wordList;
        mKeyToPosition=new HashMap<>(mWordList.size());
        for (int i=0;i<mWordList.size();++i)
            mKeyToPosition.put(mWordList.get(i).getId(),i);
    }
    @Nullable
    @Override
    public Long getKey(int position) {
        return mWordList.get(position).getId();
    }
    @Override
    public int getPosition(Long key) { return mKeyToPosition.get(key); }
}
```

# Biblioteka Room – zaznaczanie i kasowanie elementów, modyfikacje adaptera

```
public class WordListAdapter extends  
RecyclerView.Adapter<WordListAdapter.WordViewHolder> {  
//...  
private SelectionTracker<Long> mSelectionTracker;  
private WordItemKeyProvider mWordItemKeyProvider;  
public WordListAdapter(Context context,  
    WordItemKeyProvider wordItemKeyProvider) {  
    mLayoutInflater=LayoutInflater.from(context);  
    mWordItemKeyProvider=wordItemKeyProvider;  
    this.mWordList = null;  
}  
//...  
@Override  
public void onBindViewHolder(@NonNull WordViewHolder holder,  
    int position) {  
    boolean isSelected=false;  
    //w trakcie wypełniania wiersza sprawdzamy czy jest zaznaczony  
    if ((mSelectionTracker != null) && mSelectionTracker  
        .isSelected(mWordList.get(position).getId()))  
        isSelected=true;  
    holder.bindToWordViewHolder(position, isSelected);  
}
```

# Biblioteka Room – zaznaczanie i kasowanie elementów, modyfikacje adaptera

```
//...
public class WordViewHolder extends RecyclerView.ViewHolder {
    TextView wordTextView;
    //każdy wiersz ma teraz swoje szczegóły
    WordItemDetails wordItemDetails;

    public WordViewHolder(@NonNull View itemView) {
        super(itemView);
        wordTextView=itemView.findViewById(R.id.wordTextView);
        wordItemDetails=new WordItemDetails();
    }

    public void bindToWordViewHolder(int position,
                                    boolean isSelected)
    {
        wordTextView.setText(mWordList.get(position).getWord());
        wordItemDetails.id =mWordList.get(position).getId();
        wordItemDetails.position=position;
        //itemView = główny element wiersza (odziedziczony po
        //ViewHolder); aktywny == zaznaczony
        itemView.setActivated(isSelected);
    }
}
```

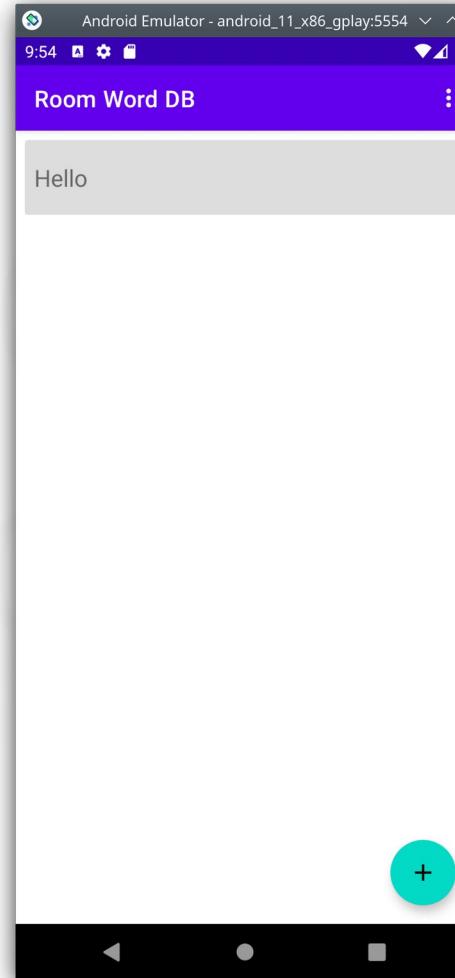
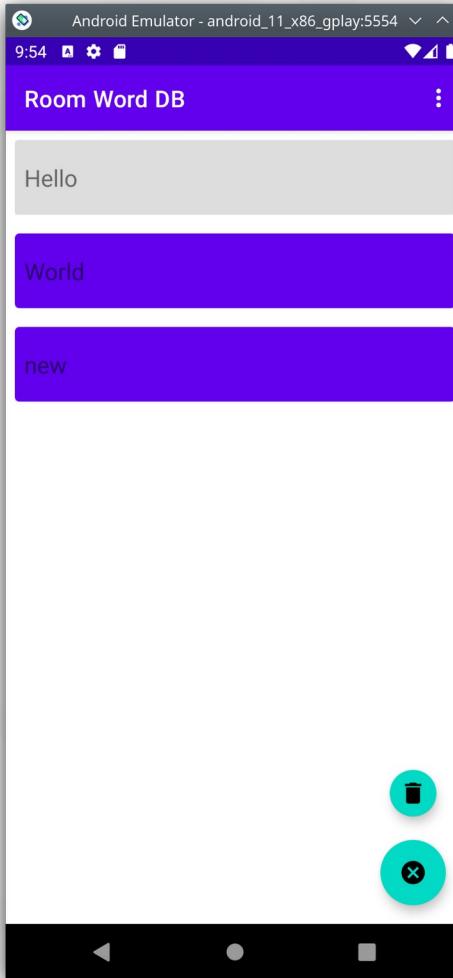
# Biblioteka Room – zaznaczanie i kasowanie elementów, modyfikacje adaptera

```
//umożliwiamy odczytanie szczegółów elementu listy (używane
//wcześniej w WordItemDetailsLookup)
public WordItemDetails getWordItemDetails() {
    return wordItemDetails;
}

public void setSelectionTracker(
    SelectionTracker<Long> mSelectionTracker) {
    this.mSelectionTracker = mSelectionTracker;
}

public void setWordList(List<Word> wordList) {
    if (mSelectionTracker!=null)
        mSelectionTracker.clearSelection();
    this.mWordList = wordList;
    mWordItemKeyProvider.setWords(wordList);
    notifyDataSetChanged();
}
```

# Biblioteka Room – zaznaczanie i kasowanie elementów, modyfikacje adaptera



# Ustawienia współdzielone – SharedPreferences

- Umożliwiają przechowywanie prostych elementów (nie nadają się do skomplikowanych struktur danych)
- Dopuszczalne typy danych: tekst, wartości liczbowe, wartości logiczne, zbiory tekstów
- Ustawienia współdzielone dostępne są dla wszystkich elementów aplikacji
- Przechowywane jako para wartości: tekstowy klucz / wartość
- Do edycji ustawień można użyć *AndroidX Preference Library* (część *Android Jetpack*). Biblioteka umożliwia:
  - Tworzenie ekranów edycji ustawień
  - Reagowanie na zmianę ustawień
  - Wybór miejsca przechowywania danych (domyślnie SharedPreferences)

# Ustawienia współdzielone – layouty

- Layout settings\_activity.xml

```
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- za frame layout zostanie wstawiony panel ustawień -->
    <FrameLayout
        android:id="@+id/settings"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

- Layout main\_activity.xml

Text preference: ?

- textPreferenceTextView

List preference: ?

- listPreferenceTextView

Switch 1 preference: ?

- switch1TextView

Switch 2 preference: ?

- switch2TextView

# Ustawienia współdzielone – specyfikacja ustawień

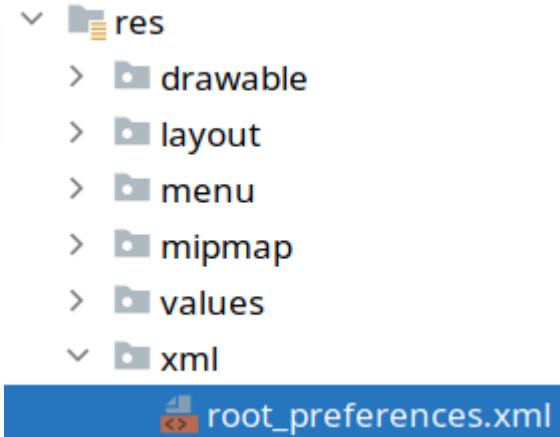
```
<PreferenceScreen xmlns:app=
    "http://schemas.android.com/apk/res-auto">

    <PreferenceCategory app:title="@string/category_1_header">

        <EditTextPreference
            app:defaultValue="default from XML"
            app:key="text_preference"
            app:title=
                "@string/text_preference_title"
            app:useSimpleSummaryProvider="true" />
        <!-- podsumowanie będzie ustawiane
            automatycznie na podstawie wartości -->

        <ListPreference
            app:entries="@array/list_entries"
            app:entryValues="@array/list_values"
            app:key="list_preference"
            app:title="@string/list_preference_title"
            app:useSimpleSummaryProvider="true" />

    </PreferenceCategory>
```



# Ustawienia współdzielone – specyfikacja ustawień

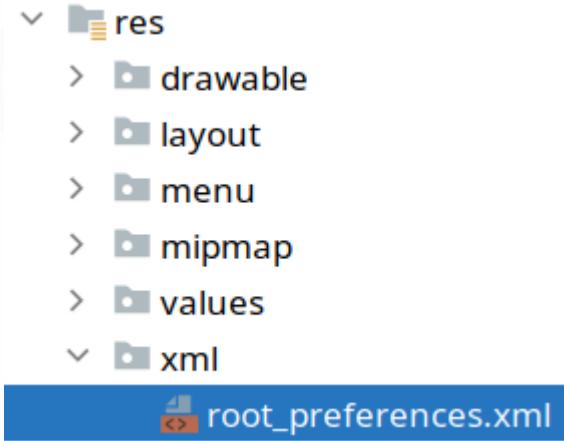
```
<PreferenceCategory app:title="@string/category_2_header">

    <SwitchPreferenceCompat
        app:defaultValue="true"
        app:key="switch_1_preference"
        app:title=
            "@string/switch_1_title" />

    <!-- switch 2 zależy od switch 1 -->
    <SwitchPreferenceCompat
        app:dependency="switch_1_preference"
        app:key="switch_2_preference"
        app:summaryOff=
            "@string/switch_2_summary_off"
        app:summaryOn=
            "@string/switch_2_summary_on"
        app:title="@string/switch_2_title" />

</PreferenceCategory>

</PreferenceScreen>
```



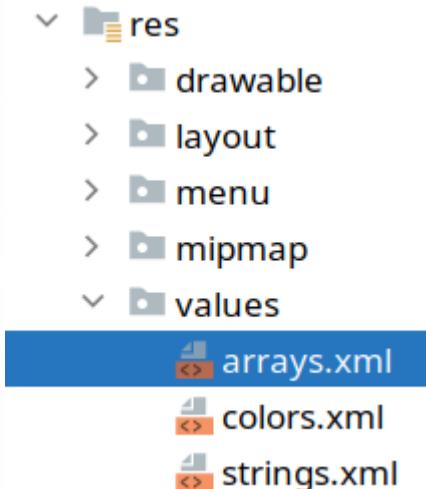
# Ustawienia współdzielone – zależności,

- build.gradle (moduł) wartości

```
dependencies {  
    //...  
    //biblioteka z obsługą ustawień  
    implementation 'androidx.preference:preference:1.1.1'  
    //..  
}
```

- arrays.xml – wartości ustawień listowych

```
<resources>  
    <!-- Reply Preference -->  
    <string-array name="list_entries">  
        <item>Entry 1</item>  
        <item>Entry 2</item>  
    </string-array>  
  
    <string-array name="list_values">  
        <item>entry_1</item>  
        <item>entry_2</item>  
    </string-array>  
</resources>
```



# Ustawienia współdzielone – aktywność ustawień

```
public class SettingsActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.settings_activity);  
        if (savedInstanceState == null) {  
            getSupportFragmentManager()  
                .beginTransaction()  
                .replace(R.id.settings, new SettingsFragment())  
                .commit();  
        }  
  
        ActionBar actionBar = getSupportActionBar();  
        if (actionBar != null) {  
            actionBar.setDisplayHomeAsUpEnabled(true);  
        }  
    }  
}
```

# Ustawienia współdzielone – aktywność ustawień

```
//fragment zdefiniowany wewnątrz aktywności
public static class SettingsFragment extends PreferenceFragmentCompat {
    @Override
    public void onCreatePreferences(Bundle savedInstanceState,
                                   String rootKey) {
        setPreferencesFromResource(R.xml.root_preferences, rootKey);
        //OnPreferenceChangeListener - ustawiany dla konkretnego
        //ustawienia
        Preference textPreference =
            getPreferenceManager().findPreference("text_preference");
        textPreference.setOnPreferenceChangeListener(
            new Preference.OnPreferenceChangeListener() {
                //uruchamiane przed zmianą ustawienia
                //(SharedPreferences lub PreferenceDataStore)
                @Override
                public boolean onPreferenceChange(
                    Preference preference, Object newValue) {
                    Toast.makeText(getContext(),
                        "Preference with key: " +
                        preference.getKey() +
                        " is about to change to: " + newValue,
                        Toast.LENGTH_SHORT).show();
                    return true; //akceptujemy zmianę ustawienia
                    //można też odrzucić
                }
            });
    }
}
```

# Ustawienia współdzielone – aktywność ustawień

```
//pola i metody fragmentu ustawień
//trzeba przechowywać silną referencję do listenera aby zapobiec
//odśmiecaniu (PreferenceManager nie przechowuje silnej referencji)
SharedPreferences.OnSharedPreferenceChangeListener listener =
    new SharedPreferences.OnSharedPreferenceChangeListener() {
        //wywoływane przed zmianą dla wszystkich ustawień po zmianie
        //ustawienia (tylko SharedPreferences)
        @Override
        public void onSharedPreferenceChanged(
            SharedPreferences sharedPreferences, String key) {
            Toast.makeText(getContext(), "Preference with key: " +
                key + " has changed", Toast.LENGTH_SHORT).show();
        }
    };
@Override
public void onResume() {
    super.onResume();
    getPreferenceManager().getSharedPreferences()
        .registerOnSharedPreferenceChangeListener(listener);
}
@Override
public void onPause() {
    super.onPause();
    getPreferenceManager().getSharedPreferences()
        .unregisterOnSharedPreferenceChangeListener(listener);
}}}//koniec fragmentu i koniec aktywności
```

# Ustawienia współdzielone – główna aktywność

```
public class MainActivity extends AppCompatActivity {  
    private static final String TAG =  
        MainActivity.class.getSimpleName();  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
    @Override  
    protected void onStart() {  
        super.onStart();  
        //uzyskanie dostępu do ustawień współdzielonych  
        SharedPreferences sharedPreferences =  
            PreferenceManager.getDefaultSharedPreferences(this);  
        //odczytanie wartości ustawienia  
        String textPreference =  
            sharedPreferences.getString("text_preference", "");  
        if (TextUtils.isEmpty(textPreference)) {  
            //modyfikacja wartości ustawień  
            SharedPreferences.Editor editor=sharedPreferences.edit();  
            editor.putString("text_preference","default from Java");  
            editor.commit();  
        }  
    }  
}
```

# Ustawienia współdzielone – główna aktywność

```
//odczytanie i wyświetlenie wartości ustawień
TextView textPreferenceTextView =
    findViewById(R.id.textPreferenceTextView);
textPreferenceTextView.setText(
    sharedPreferences.getString("text_preference","?"));

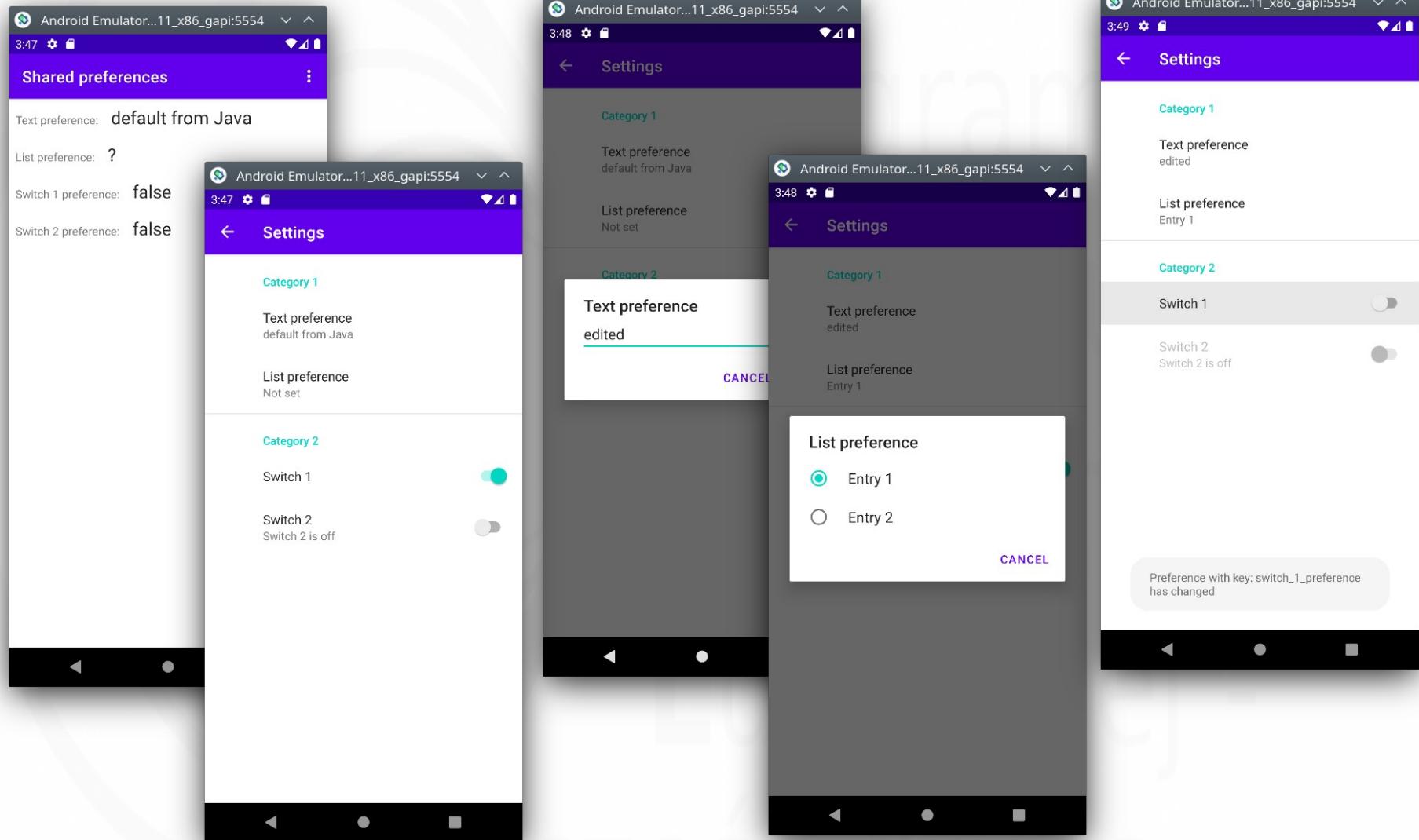
TextView listPreferenceTextView =
    findViewById(R.id.listPreferenceTextView);
listPreferenceTextView.setText(
    sharedPreferences.getString("list_preference","?"));

TextView switch1TextView = findViewById(R.id.switch1TextView);
switch1TextView.setText(Boolean.toString(
    sharedPreferences.getBoolean("switch_1_preference",false)));

TextView switch2TextView = findViewById(R.id.switch2TextView);
switch2TextView.setText(Boolean.toString(
    sharedPreferences.getBoolean("switch_2_preference",false)));
}

//tworzenie i obsługa opcji menu - standardowa...
}
```

# Ustawienia współdzielone





## Programowanie Aplikacji Mobilnych na Platformę Android

### Wykład 6 – Wielozadaniowość na platformie Android. Komunikacja składników aplikacji.

Jakub Smołka



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny

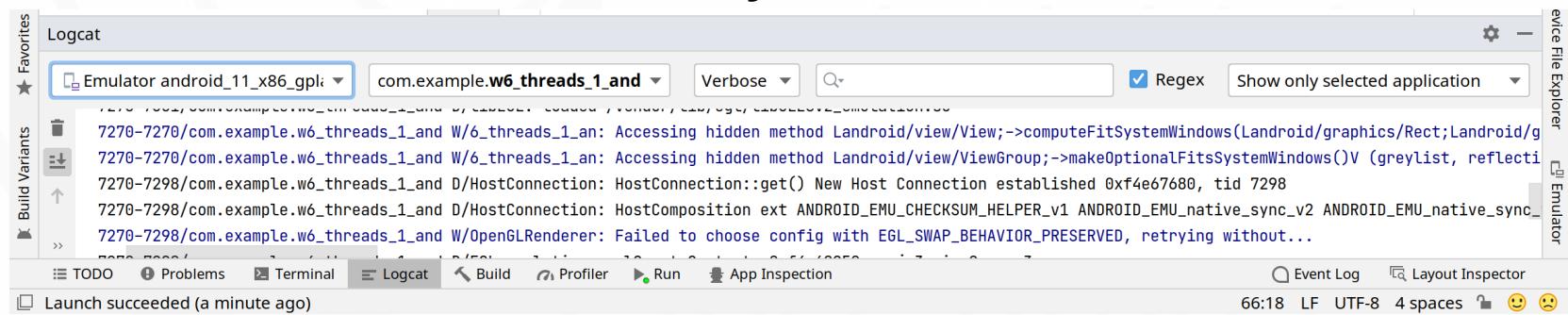


# Wielozadaniowość – wprowadzenie

- W Androidzie nie należy wykonywać zadań, długotrwałych w głównym wątku aplikacji (on obsługuje interfejs użytkownika)
- Gdy główny wątek jest zajęty aplikacja nie reaguje na polecenia użytkownika i
  - w starszych wersjach pojawiał się tzw. komunikat *ANR* (*ang. application not responding*)
  - w nowszych wersjach Aplikacja zgłasza wyjątek i przerywa wykonanie (chodzi o zmuszenie programistów do poprawienia aplikacji)
- Przykładem operacji, których nie wolno wykonywać w wątku głównym są operacje sieciowe

# Klasa Log – wyświetlanie informacji z wątków

- Klasa Log pozwala na wysyłanie komunikatów do LogCat
- Posiada metody statyczne:
  - v() - wysyłanie szczegółowych informacji
  - d() - wysyłanie informacji służących do debugowania
  - i() - wysyłanie informacji
  - w() - wysyłanie ostrzeżeń
- Parametrami najprostszych wersji ww. metod są: etykieta tekstowa i komunikat tekstowy



# Wątki Javy – klasa Thread

- Stanowi standardowy sposób obsługi wątków wbudowany w język Java (nie tylko w Androida)
- Jest wykorzystywany pośrednio we wszystkich kolejnych omówionych metodach wykonywania zadań w tle
- Istnieją dwa sposoby utworzenia wątku w Javie:
  - utworzenie klasy pochodnej dziedziczącej po `Thread()` i definiującej własną metodę `run()`
  - utworzenie klasy implementującej interfejs `Runnable`, którego częścią jest metoda `run()`. A następnie przekazanie nowego obiektu tej klasy jako parametr konstruktora klasy `Thread`. Zamiast takiego obiektu można również użyć wyrażenia lambda
- Niezależnie od wybranego sposobu metoda `run()` powinna zawierać warunki, które spowodują, że kiedyś się zakończy

# Klasa Handler

- Klasa frameworku Androida, która rozwiązuje problem przesyłania danych między wątkami wykonującymi pewne zadania a wątkiem GUI
- Obiekt klasy Handler jest związany z wątkiem, którego Looper otrzymał jako parametr konstruktora
- Looper to obiekt realizujący obsługę pętli komunikatów w wątku – w tym wątku Handler przetwarza komunikaty
- Zapewnienie przesyłania komunikatów wymaga:
  - stworzenia klasy pochodnej klasy Handler i zdefiniowania w niej metody `handleMessage()` lub
  - utworzenia klasy implementującej `Runnable` (lub wyrażenia lambda) i zastosowanie metody `post()` handlera

# Współpraca Thread i Handler

## Wątek GUI

- Handler
  - reakcja na komunikat np. aktualizacja GUI
  - wykonanie obiektu Runnable

## Wątek zadania

- metoda run() wywołuje Handler.sendMessage() i wysyła obiekt Message do handlera
- ewentualnie może wywoływać metodę Handler.post() i wysłać obiekt Runnable do wykonania przez handler

# Wątki 1 – layout, główna aktywność

Time:

0

START

- Etykieta tekstowa.  
Identyfikator:
  - time\_label
- Przycisk. Identyfikator:
  - start\_button

# Wątki 1 – główna aktywność

```
public class MainActivity extends AppCompatActivity {  
  
    private static final String TAG =  
        MainActivity.class.getSimpleName();  
  
    //handler służący do przesyłania komunikatów  
    private Handler mHandler;  
    private TextView mTimeLabel;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        // etykieta, którą będziemy aktualizować za pomocą handlera  
        mTimeLabel = findViewById(R.id.time_text_view);  
  
        //obsługa przycisku uruchamiającego odliczanie  
        Button startButton = findViewById(R.id.start_button);  
        startButton.setOnClickListener(v -> startThread());  
    }  
}
```

# Wątki 1 – główna aktywność, metoda handleMessage()

```
private void startThread() {  
    // Handler będzie przekazywał komunikaty do wątku GUI  
    // jest związany z wątkiem, w którego Looper otrzyma  
    mHandler = new Handler(Looper.getMainLooper()) {  
        //metoda odpowiedzialna za odebranie komunikatu  
        @Override  
        //obiekt Message zawiera komunikat  
        public void handleMessage(Message msg) {  
            Log.d(TAG, "handler received: " + msg.what  
                  + " from thread");  
            mTimeLabel.setText(Integer.toString(msg.what));  
            super.handleMessage(msg);  
        }  
    };  
  
    // tworzenie wątku  
    Thread thread = new Thread() {  
        @Override  
        public void run() {  
            for (int time = 0; time <= 10; time++) {
```

# Wątki 1 – główna aktywność, wysyłanie komunikatów

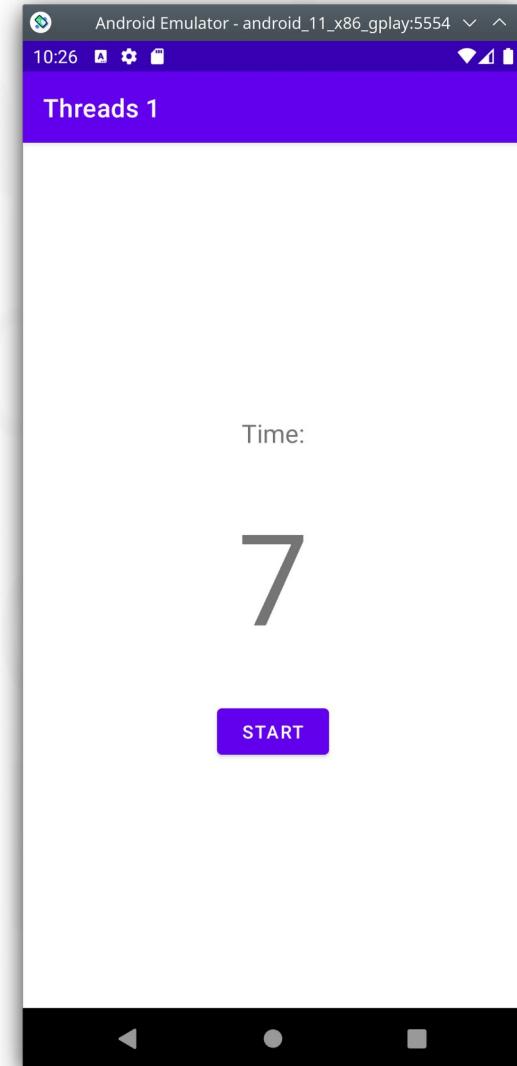
```
// wysyła komunikat (obiekt Message) tylko z
//polem what
Log.d(TAG, "handler sent: " + time + " to GUI");
mHandler.sendMessage(time);
// (można też wysłać obiekt Message zawierający
//więcej danych
if (time == 10) return;
try {
    // udaję, że pracuję ;-)
    Thread.sleep(1000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
}
};

thread.start();
Log.d(TAG, "thread started");
}
```

# Wątki 1

- LogCat

```
2312-2312/com.example.w6_threads_1_and D/MainActivity: thread started  
2312-3139/com.example.w6_threads_1_and D/MainActivity: handler sent: 0 to GUI  
2312-2312/com.example.w6_threads_1_and D/MainActivity: handler received: 0 from thread  
2312-3139/com.example.w6_threads_1_and D/MainActivity: handler sent: 1 to GUI  
2312-2312/com.example.w6_threads_1_and D/MainActivity: handler received: 1 from thread  
2312-3139/com.example.w6_threads_1_and D/MainActivity: handler sent: 2 to GUI  
2312-2312/com.example.w6_threads_1_and D/MainActivity: handler received: 2 from thread  
2312-3139/com.example.w6_threads_1_and D/MainActivity: handler sent: 3 to GUI  
2312-2312/com.example.w6_threads_1_and D/MainActivity: handler received: 3 from thread
```



# Wątki 2 – główna aktywność

```
public class MainActivity extends AppCompatActivity {

    private final static String TAG =
        MainActivity.class.getSimpleName();
    private Handler mHandler;
    private TextView mTimeLabel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Handler będzie przekazywał komunikaty do wątku GUI
        // wykonuje zadania w wątku, którego Looper dostanie
        mHandler = new Handler(Looper.getMainLooper());
        // etykieta, którą będziemy aktualizować za pomocą handlera
        mTimeLabel = findViewById(R.id.time_text_view);

        // obsługa kliknięcia przycisku
        // ...
        Button startButton = findViewById(R.id.start_button);
        startButton.setOnClickListener(v -> startThread());
    }
}
```

# Wątki 2 – główna aktywność, wysyłanie zadań

```
private void startThread() {  
    // tworzenie wątku  
    Thread thread = new Thread(new Task());  
    thread.start();  
    Log.d(TAG, "thread started");  
}  
  
class Task implements Runnable {  
    private void send(int time) {  
        Log.d(TAG, "thread sent: " + time + " to GUI");  
        // zmienna lokalna do której odwołuje się klasa wewnętrzna  
        // musi być final  
        final int finalTime = time;  
        // metoda post wymaga obiektu z metodą run() - zadaniem do  
        // wykonania wykonywane w wątku hanldera - tu w wątku GUI  
        mHandler.post(() -> {  
            Log.d(TAG, "handler received: " + finalTime  
                + " from thread");  
            mTimeLabel.setText(Integer.toString(finalTime));  
        });  
    }  
}
```

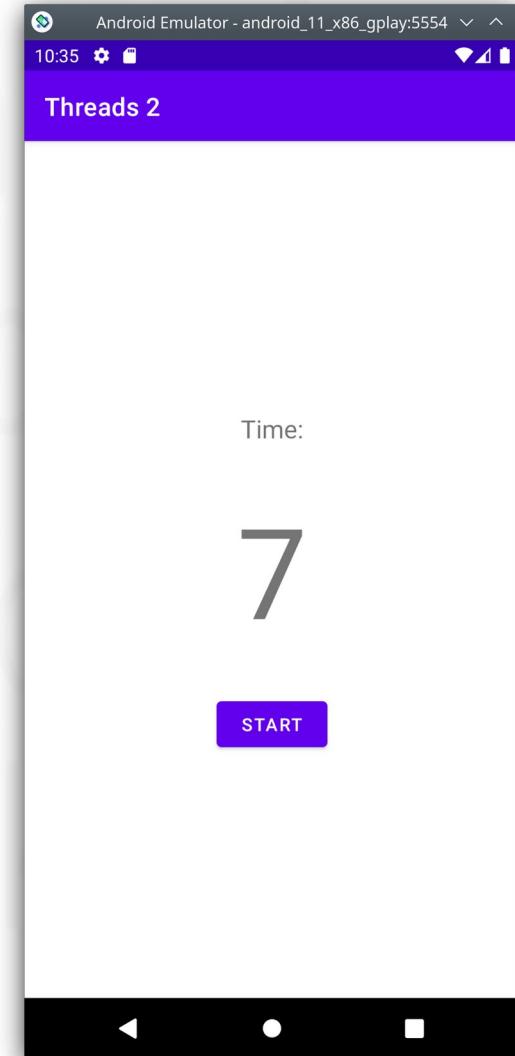
# Wątki 2 – główna aktywność, wysyłanie zadań

```
@Override  
public void run() {  
    for (int time = 0; time <= 10; time++) {  
        // wysyłanie informacji o postępach  
        send(time);  
        if (time==10) return;  
        try {  
            // udaje, że pracuję ;-)  
            Thread.sleep(1000);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

# Wątki 2

- LogCat

```
5149-5149/com.example.w6_threads_2_and D/MainActivity: thread started  
5149-5178/com.example.w6_threads_2_and D/MainActivity: thread sent: 0 to GUI  
5149-5149/com.example.w6_threads_2_and D/MainActivity: handler received: 0 from thread  
5149-5178/com.example.w6_threads_2_and D/MainActivity: thread sent: 1 to GUI  
5149-5149/com.example.w6_threads_2_and D/MainActivity: handler received: 1 from thread  
5149-5178/com.example.w6_threads_2_and D/MainActivity: thread sent: 2 to GUI  
5149-5149/com.example.w6_threads_2_and D/MainActivity: handler received: 2 from thread  
5149-5178/com.example.w6_threads_2_and D/MainActivity: thread sent: 3 to GUI  
5149-5149/com.example.w6_threads_2_and D/MainActivity: handler received: 3 from thread
```



# Usługi w systemie Android

- Tworzenie usług wymaga wykorzystania mechanizmów specyficznych dla platformy
- W Android SDK klasa `Service` i jej pochodne pozwalają na osiągnięcie prawdziwej wielozadaniowości
- Usługi reprezentują długotrwałe czynności np. odtwarzanie muzyki, przesyłanie danych przez sieć, świadczenie usług na rzecz innych aplikacji
- Domyślnie klasy/usługi dziedziczące po klasie `Service` nie uruchamiają własnych wątków czy procesów i działają w głównym wątku aplikacji. W przypadku operacji, które mogłyby zablokować główny wątek na dłużej (np. operacje sieciowe) usługa powinna uruchomić swój własny wątek

# Usługi – główna aktywność, uruchamianie usługi

```
public class MainActivity extends AppCompatActivity {  
    private TextView mTimeTextView;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        Button startServiceButton = findViewById(R.id.start_button);  
        startServiceButton.setOnClickListener(v -> startMyService());  
  
        mTimeTextView = findViewById(R.id.time_text_view);  
    }  
    private static final String TAG =  
        MainActivity.class.getSimpleName();  
  
    public void startMyService() {  
        Log.d(TAG, "startMyService() - starting service");  
        Intent serviceIntent = new Intent(this, MyService.class);  
        startService(serviceIntent);  
        Log.d(TAG, "startMyService() - service started");  
    }  
}
```

# Rozgłoszenia

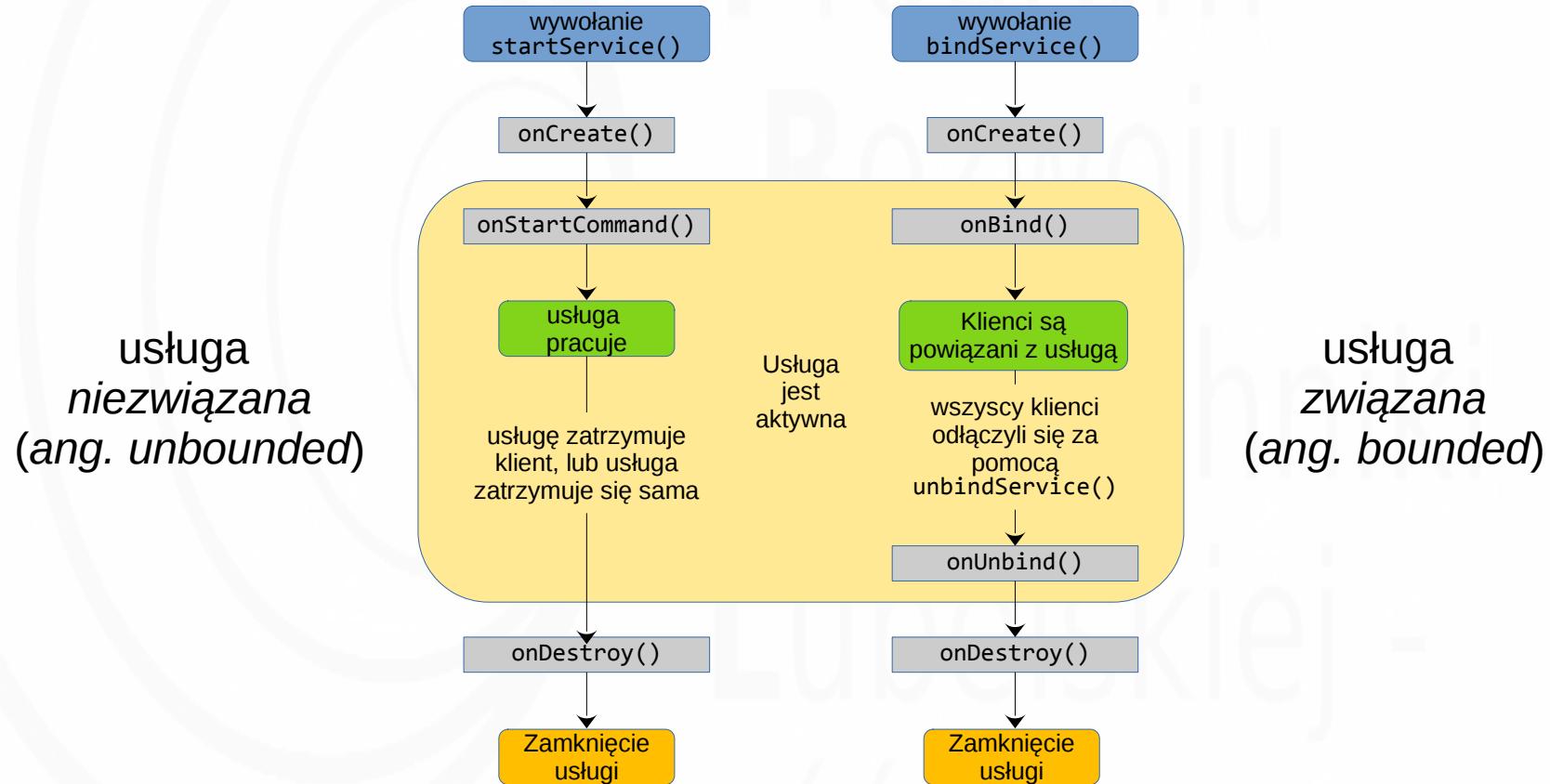
- W wielu aplikacjach dla systemu Android składniki muszą się ze sobą komunikować
- *Rozgłoszenia i odbiorcy rozgłoszeń* umożliwiają:
  - przesyłanie danych pomiędzy składnikami jednej aplikacji
  - reagowanie na zdarzenia systemowe takie jak: zakończenie ładowania systemu, zmiana stanu akumulatora, itp.
  - Istnieją dwa typy rozgłoszeń: globalne i lokalne
  - Odbieranie rozgłoszeń wymaga: stworzenia odbiorcy (zastąpiona metoda `onReceive()`), zadbanie o rejestrację i wyrejestrowanie odbiorcy w metodach `onResume()` / `onPause()` aktywności
  - Odbiorca rozgłoszeń (a w zasadzie jego metoda `onReceive()`) powinien szybko zakończyć swoje działanie
  - Możliwe jest tworzenie własnych rozgłoszeń

# Usługi – główna aktywność, rozgłoszenia

```
//komunikacja z usługą
class TimeBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        int time = intent.getIntExtra(MyService.TIME_EXTRA, -1);
        Log.d(TAG, "onReceive() - time received by broadcast: "
                + time);
        mTimeTextView.setText(Integer.toString(time));
    }
    private TimeBroadcastReceiver mTimeBroadcastReceiver
            = new TimeBroadcastReceiver();
    @Override
    protected void onStart() {
        super.onStart();
        registerReceiver(mTimeBroadcastReceiver,
                new IntentFilter(MyService.ACTION_BROADCAST));
    }
    @Override
    protected void onStop() {
        unregisterReceiver(mTimeBroadcastReceiver);
        super.onStop();
    }
}
```

# Cykl życia usługi / typy usług

- Istnieją dwa podstawowe typy usług (*unbounded* i *bounded*)
- Wiązanie (ang. *binding*) z usługą umożliwia wywoływanie jej metod



źródło: <https://developer.android.com/guide/components/services>

# Powiadomienia

- Powiadomienia wyświetlane są na pasku znajdującym się u góry ekranu
- Powiadomienia te są bardzo dobrym miejscem do wyświetlania informacji przez usługi pracujące w tle
- Powiadomienia wyświetlane są niezależnie od tego, czy aktywność aplikacji jest widoczna czy też nie
- Poprzez kliknięcie powiadomienia można powrócić do aktywności
- Najprostszym sposobem tworzenia powiadomień jest wykorzystanie klasy `Notification.Builder`

# Usługi pierwszoplanowe

- W nowszych wersjach Androida należy tworzyć tylko usługi pierwszoplanowe
- Usługa pierwszoplanowa to taka, której działania użytkownik jest świadom – warunkiem działania takiej usługi jest wyświetlanie przez nią powiadomienia
- Jeżeli usługa nie jest pierwszoplanowa to zostanie zakończona kilkanaście sekund po tym jak aplikacja przestanie być widoczna na ekranie
- Tworzenie usług pierwszoplanowych wymaga dodania uprawnienia do pliku `AndroidManifest.xml`
- Usługi muszą być też wymienione w manifeście (podobnie jak aktywności)

# Usługi – manifest aplikacji

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.w6_service_and">

    <uses-permission android:name=
        "android.permission.FOREGROUND_SERVICE"/>

    <application
        <!-- ... -->
        <service
            android:name=".MyService"
            android:enabled="true"
            android:exported="true"></service>
    </application>

</manifest>
```

# Usługi – definicje stałych w usłudze

```
public class MyService extends Service {  
  
    private static final String TAG =  
        MyService.class.getSimpleName();  
    private static final String NOTIFICATION_CHANNEL_ID =  
        "com.example.service.notification_channel";  
    private static final String NOTIFICATION_CHANNEL_NAME =  
        "com.example.service.notification_channel";  
    private static final int NOTIFICATION_ID = 1;  
  
    public static final String ACTION_BROADCAST =  
        "com.example.service.broadcast";  
    public static final String TIME_EXTRA =  
        "com.example.service.broadcast.time";  
  
    private NotificationManager mNotificationManager;  
  
    private HandlerThread mServiceHandlerThread;  
    private Handler mServiceThreadHandler;  
    private Handler mMainThreadHandler;  
  
    public MyService() { }
```

# Usługi – wątek, kanał powiadomień

```
//wykonuje jednorazową konfigurację - niezależnie od tego czy
//usługę uruchomiono za pomocą startService() czy bindService()
@Override
public void onCreate() {
    super.onCreate();
    Log.d(TAG, "onCreate()");
    mServiceHandlerThread = new HandlerThread(TAG);
    mServiceHandlerThread.start();
    mServiceThreadHandler =
        new Handler(mServiceHandlerThread.getLooper());
    mMainThreadHandler = new Handler(Looper.getMainLooper());
    mNotificationManager = (NotificationManager)
        getSystemService(NOTIFICATION_SERVICE);
    // Android O wymaga kanału powiadomień
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        // utwórz kanał dla powiadomienia
        NotificationChannel mChannel =
            new NotificationChannel(NOTIFICATION_CHANNEL_ID,
                NOTIFICATION_CHANNEL_NAME,
                NotificationManager.IMPORTANCE_LOW);
        // utwórz kanał powiadomień w menedżerze powiadomień
        mNotificationManager.createNotificationChannel(mChannel);
    }
}
```

# Usługi – cykl życia usługi

```
@Override  
public void onDestroy() {  
    super.onDestroy();  
    Log.d(TAG, "onDestroy()");  
}  
  
//trzeba zaimplementować (w najprostszym przypadku zwraca null -  
//unbound service)  
@Override  
public IBinder onBind(Intent intent) {  
    Log.d(TAG, "onBind()");  
    return null;  
}  
  
//wywoływana po wywołaniu startService()  
@Override  
public int onStartCommand(Intent intent, int flags, int startId)  
{  
    Log.d(TAG, "onStartCommand()");  
    startTimer();  
    return super.onStartCommand(intent, flags, startId);  
}
```

# Usługi – wysyłanie rozgłoszeń

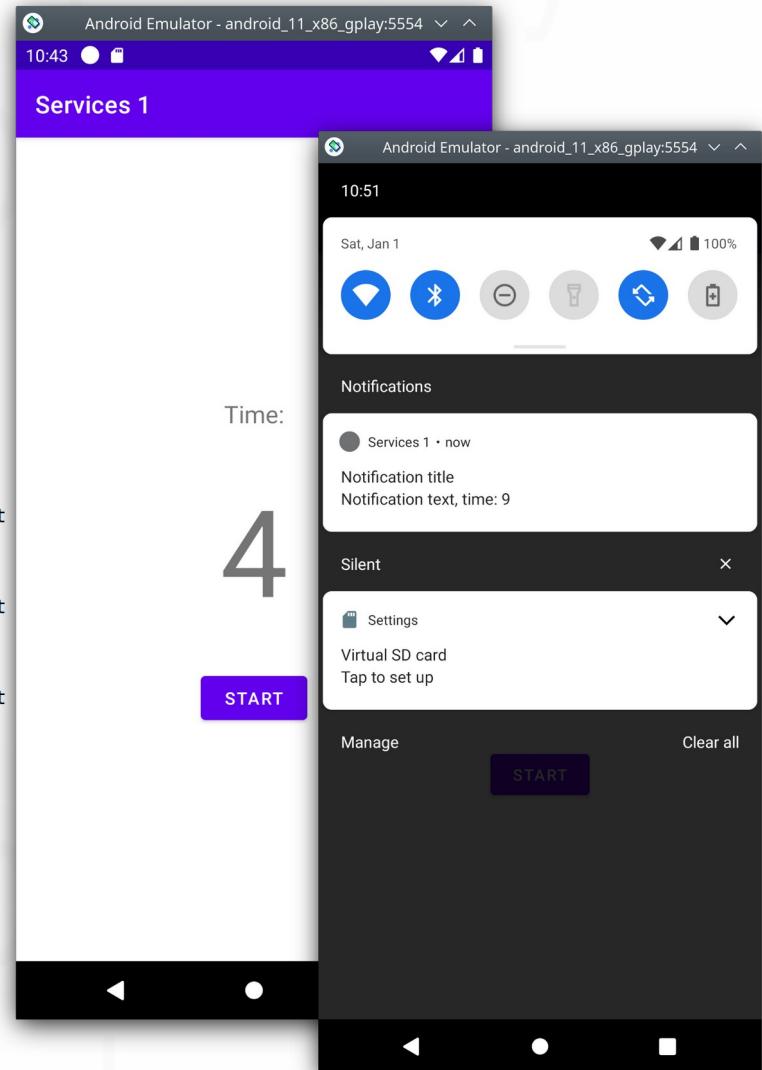
```
public void startTimer() {  
    Log.d(TAG, "startTimer() - starting foreground");  
    startForeground(NOTIFICATION_ID, getNotification(0));  
    mServiceThreadHandler.post(() -> {  
        for (int time = 0; time <= 30; time++) {  
            Log.d(TAG, "startTimer()/lambda - current time: " +  
                time);  
            updateNotification(time);  
            sendTimeBroadcast(time);  
            try {  
                Thread.sleep(1000);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
        mMainThreadHandler.post(() -> stopSelf());  
    });  
}  
//przesyłanie danych  
private void sendTimeBroadcast(int time) {  
    Intent broadcastIntent = new Intent(ACTION_BROADCAST);  
    broadcastIntent.putExtra(TIME_EXTRA, time);  
    sendBroadcast(broadcastIntent);  
    Log.d(TAG, "sendTimeBroadcast() - sent time: " + time +  
        " using broadcast");  
}
```

# Usługi – wyświetlanie powiadomień

```
private Notification getNotification(int time) {  
    Intent intent = new Intent(this, MyService.class);  
  
    NotificationCompat.Builder builder =  
        new NotificationCompat.Builder(this,  
            NOTIFICATION_CHANNEL_ID)  
        .setContentText("Notification text, time: " + time)  
        .setContentTitle("Notification title")  
        .setOngoing(true)  
        .setPriority(Notification.PRIORITY_HIGH)  
        .setSmallIcon(R.mipmap.ic_launcher)  
        .setTicker("Notification ticker")  
        .setWhen(System.currentTimeMillis());  
    return builder.build();  
}  
  
private void updateNotification(int time) {  
    mNotificationManager.notify(NOTIFICATION_ID,  
        getNotification(time));  
}  
}
```

# Usługi – wyświetlanie powiadomień

```
5653-5653/com.example.w6_service_and D/MainActivity: startMyService() - starting service  
5653-5653/com.example.w6_service_and D/MainActivity: startMyService() - service started  
5653-5653/com.example.w6_service_and D/MyService: onCreate()  
5653-5653/com.example.w6_service_and D/MyService: onStartCommand()  
5653-5653/com.example.w6_service_and D/MyService: startTimer() - starting foreground  
5653-5681/com.example.w6_service_and D/MyService: startTimer() /lambda - current time: 0  
5653-5681/com.example.w6_service_and D/MyService: sendTimeBroadcast() - sent time: 0 using broadcast  
5653-5653/com.example.w6_service_and D/MainActivity: onReceive() - time received by broadcast: 0  
5653-5681/com.example.w6_service_and D/MyService: startTimer() /lambda - current time: 1  
5653-5681/com.example.w6_service_and D/MyService: sendTimeBroadcast() - sent time: 1 using broadcast  
5653-5653/com.example.w6_service_and D/MainActivity: onReceive() - time received by broadcast: 1  
5653-5681/com.example.w6_service_and D/MyService: startTimer() /lambda - current time: 2  
5653-5681/com.example.w6_service_and D/MyService: sendTimeBroadcast() - sent time: 2 using broadcast  
5653-5653/com.example.w6_service_and D/MainActivity: onReceive() - time received by broadcast: 2
```



# Obiekty LiveData

- LiveData to pojemnik na dane, który może być obserwowany w jakimś cyklu życia
- Do obiektu LiveData za pomocą observe() można dodać parę: obserwator i właściciel cyklu życia (LifecycleOwner). Obserwator będzie powiadamiany o zmianach danych gdy powiązany właściciel cyklu życia jest aktywny (stan: STARTED lub RESUMED)
- Obserwator dodany przez observeForever() jest uważany za zawsze aktywnego
- Obserwatora można usunąć za pomocą removeObserver()
- Obserwator dodany razem z cyklem życia zostanie automatycznie usunięty, jeśli odpowiadający mu cykl życia przejdzie do stanu DESTROYED
- metody onActive() i onInactive() umożliwiają powiadamianie gdy liczba obserwatorów zmienia się  $0 \leftrightarrow 1$
- Klasa LiveData jest przeznaczona do przechowywania pól danych ViewModel, ale może być również używana do przekazywania danych między różnymi elementami aplikacji.

# Usługi – główna aktywność, wiązanie z usługą

```
public class MainActivity extends AppCompatActivity {  
  
    private TextView mTimeTextView;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        Intent bindIntent = new Intent(this, MyService.class);  
        bindService(bindIntent, mServiceConnection,  
                   Context.BIND_AUTO_CREATE);  
  
        Button startServiceButton = findViewById(R.id.start_button);  
        startServiceButton.setOnClickListener(v -> startMyService());  
  
        mTimeTextView = findViewById(R.id.time_text_view);  
    }  
}
```

# Usługi – główna aktywność, wiązanie z usługą

```
private static final String TAG =
    MainActivity.class.getSimpleName();

boolean mServiceConnected = false;
MyService.MyServiceBinder mMyServiceBinder = null;

public void startMyService() {
    Log.d(TAG, "startMyService() - binding to service");
    //inaczej uruchamiamy usługę (bez startService)
    if (mServiceConnected)
        mMyServiceBinder.getService().startTimer();
}

@Override
protected void onDestroy() {
    super.onDestroy();
    unbindService(mServiceConnection);
}
```

# Usługi – główna aktywność, wiązanie z usługą

```
ServiceConnection mServiceConnection = new ServiceConnection() {  
    @Override  
    public void onServiceConnected(ComponentName componentName,  
                                   IBinder iBinder) {  
        mServiceConnected = true;  
        mMyServiceBinder = (MyService.MyServiceBinder) iBinder;  
        mMyServiceBinder.getService()  
            .getCurrentTimeLiveData().observe(  
                MainActivity.this, time -> {  
                    Log.d(TAG, "observe()/lambda - Time received"  
                        + "through live data: " + time);  
  
        mTimeTextView.setText(Integer.toString(time));  
    });}  
    @Override  
    public void onServiceDisconnected(ComponentName  
componentName) {  
        mServiceConnected = false;  
        mMyServiceBinder = null;  
        mMyServiceBinder.getService()  
            .getCurrentTimeLiveData()  
            .removeObservers(MainActivity.this);  
    }};}
```

# Usługi – binder

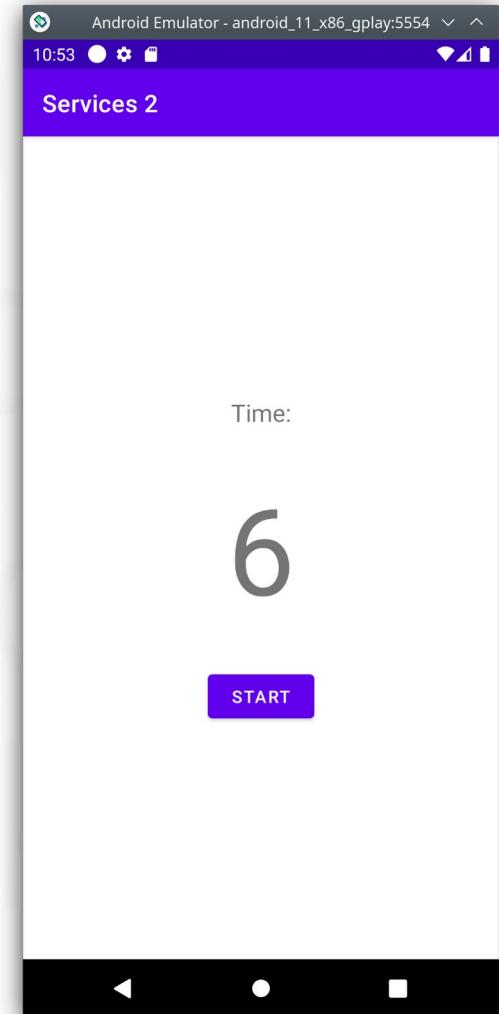
```
public class MyService extends Service {  
    //powiadomienia identyczne...  
    //wątki, handlery identyczne...  
  
    //binder umożliwia wywoływanie metod usługi z komponentów  
    //aplikacji  
    private final IBinder mBinder = new MyServiceBinder();  
    //LiveData jest abstrakcyjna  
    private MutableLiveData<Integer> mCurrentTimeLiveData;  
  
    public class MyServiceBinder extends Binder {  
        public MyService getService() {  
            return MyService.this;  
        }  
    }  
    public MyService() { }  
    @Override  
    public void onCreate() {  
        //...  
        //tutaj jedyną różnicą jest utworzenie LiveData do  
        //komunikacji  
        mCurrentTimeLiveData = new MutableLiveData<>();  
    }  
}
```

# Usługi – binder, LiveData

```
@Override
public IBinder onBind(Intent intent) {
    Log.d(TAG, "onBind()");
    return mBinder; //teraz zwracamy binder
}
public void startTimer() {
    Log.d(TAG, "startTimer() - starting foreground");
    startForeground(NOTIFICATION_ID, getNotification(0));
    mServiceThreadHandler.post(() -> {
        for (int time = 0; time <= 30; time++) {
            Log.d(TAG, "startTimer()/lambda - current time: " +
                  time);
            updateNotification(time);
            //postValue bo z innego wątku
            mcurrentTimeLiveData.postValue(time);
            Log.d(TAG, "startTimer()/lambda - sent time: " +
                  time + " using LiveData");
            try { Thread.sleep(1000); }
            catch (InterruptedException e) { e.printStackTrace(); }
        }
    })
    mMainThreadHandler.post(() -> stopSelf());
}); }
public LiveData<Integer> getCurrentTimeLiveData() {
    return mcurrentTimeLiveData;
}
}
```

# Usługi – binder i LiveData

```
5766-5766/com.example.w6_service_and D/MainActivity: startMyService() - binding to service
5766-5766/com.example.w6_service_and D/MyService: startTimer() - starting foreground
5766-5792/com.example.w6_service_and D/MyService: startTimer()/Lambda - current time: 0
5766-5792/com.example.w6_service_and D/MyService: startTimer()/lambda - sent time: 0 using LiveData
5766-5766/com.example.w6_service_and D/MainActivity: observe()/Lambda - Time received through live data: 0
5766-5792/com.example.w6_service_and D/MyService: startTimer()/Lambda - current time: 1
5766-5792/com.example.w6_service_and D/MyService: startTimer()/Lambda - sent time: 1 using LiveData
5766-5766/com.example.w6_service_and D/MainActivity: observe()/Lambda - Time received through live data: 1
5766-5792/com.example.w6_service_and D/MyService: startTimer()/Lambda - current time: 2
5766-5792/com.example.w6_service_and D/MyService: startTimer()/Lambda - sent time: 2 using LiveData
5766-5766/com.example.w6_service_and D/MainActivity: observe()/lambda - Time received through live data: 2
```





# Programowanie Aplikacji Mobilnych na Platformę Android

## Wykład 7 – Podstawy komunikacji sieciowej na platformie Android.

Jakub Smołka



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



# Sieć emulatora Android

- Każdy emulator jest „podłączony” do wirtualnego routera / firewalla
- Izoluje on emulator od sieci komputera, na którym pracuje
- Wirtualny router tworzy sieć 10.0.2.0/24, w której adresy przypisane są następująco

| Adres(y)       | Opis   |
|----------------|--|
| 10.0.2.1       | Adres wirtualnego routera                              |
| 10.0.2.2       | Adres komputera (alias dla adresu 127.0.0.1 komputera) |
| 10.0.2.3       | Pierwszy serwer DNS                                    |
| 10.0.2.4 do .6 | Drugi, trzeci i czwarty serwer DNS (opcjonalnie)       |
| 10.0.2.15      | Adres emulowanego urządzenia                           |
| 127.0.0.1      | Adres loopback emulowanego urządzenia                  |

# Usługi REST

- Znaczna część połączeń sieciowych realizowanych przez aplikacje mobilne do połączenia z usługami stworzonymi w stylu REST
- Na kolejnych slajdach (przed klientem) przedstawiono prostą usługę – bazę słów stworzoną z wykorzystaniem:
  - *Spring Boot* – wstępnie skonfigurowane rozwiązanie pozwalające na tworzenie aplikacji we frameworku Spring. Aplikacje zawierają wbudowany serwer aplikacji
  - *Bazy H2* – otwartoźródłowy RDMS napisany w Javie. Posiada interfejs JDBS
  - *Swagger* – narzędzie do dokumentowania sieciowych API

# Prosta usługa REST (baza słów) – pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.6</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.example</groupId>
  <artifactId>words_service</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>w7_words_service</name>
  <description>Word database</description>
  <properties>
    <java.version>11</java.version>
  </properties>
```

# Prosta usługa REST (baza słów) – pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>javax.persistence</groupId>
        <artifactId>javax.persistence-api</artifactId>
    </dependency>
    <dependency>
        <groupId>javax.validation</groupId>
        <artifactId>validation-api</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-swagger2</artifactId>
        <version>2.9.2</version>
        <type>jar</type>
    </dependency>
```

# Prosta usługa REST (baza słów) – pom.xml

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>
```

# Prosta usługa REST (baza słów) – application.properties

```
#aby wykonać plik sql
spring.jpa.hibernate.ddl-auto=none
#lokalizacja bazy
spring.datasource.url=jdbc:h2:mem:testdb
#w pliku
#spring.datasource.url = jdbc:h2:file:/tmp/words.h2
spring.datasource.driverClassName=org.h2.Driver
#domyślny użytkownik
spring.datasource.username=sa
spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

#http://localhost:8080/h2-console - konsola do zarządzania bazą
#jeżeli nie działa 127.0.0.2 nazwaHosta nazwaHosta -> /etc/hosts
spring.h2.console.enabled=true
spring.h2.console.settings.web-allow-others=false
```

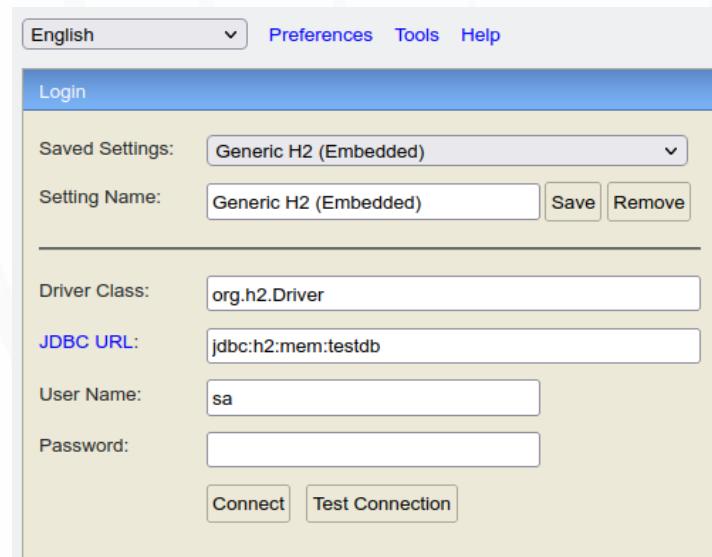
# Prosta usługa REST (baza słów) – data.sql

```
DROP TABLE IF EXISTS words;
```

```
CREATE TABLE words(id INT AUTO_INCREMENT PRIMARY KEY, word  
VARCHAR(255) NOT NULL);
```

```
INSERT INTO words (id, word) VALUES (1, 'Spring');  
INSERT INTO words (id, word) VALUES (2, 'Boot');  
INSERT INTO words (id, word) VALUES (3, 'service');
```

<http://localhost:8080/h2-console>



The screenshot shows the H2 Console database browser and a query results panel. On the left, the database tree shows 'jdb:h2:mem:testdb' expanded, revealing 'WORDS', 'INFORMATION\_SCHEMA', 'Sequences', and 'Users'. A tooltip indicates 'H2 1.4.200 (2019-10-14)'. On the right, a SQL statement 'SELECT \* FROM WORDS;' is entered in the 'SQL statement:' field. Below it, the results of the query are displayed in a table:

| ID | WORD    |
|----|---------|
| 1  | Spring  |
| 2  | Boot    |
| 3  | service |

Text at the bottom of the results panel says '(3 rows, 3 ms)' and there is an 'Edit' button.

# Prosta usługa REST (baza słów) – Swagger2Config.java

```
@Configuration  
@EnableSwagger2  
public class Swagger2Config {  
  
    @Bean  
    public Docket api() {  
        return new Docket(DocumentationType.SWAGGER_2).select()  
            .apis(RequestHandlerSelectors  
                  .basePackage("com.example.words_service"))  
            .paths(PathSelectors.regex("/.*"))  
            .build().apiInfo(apiEndPointsInfo());  
    }  
  
    private ApiInfo apiEndPointsInfo() {  
        return new ApiInfoBuilder().title("words API")  
            .description("API for word dictionary")  
            .version("0.0.1")  
            .build();  
    }  
}
```

# Prosta usługa REST (baza słów) – WordEntity.java

```
@Entity
@Table(name = "words")
public class WordEntity implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public Integer id;

    @Basic(optional = false)
    @NotNull
    public String word;

    @Override
    public String toString() {
        return "WordEntity{" + "id=" + id + ", word=" + word + '}';
    }
}
```

# Prosta usługa REST (baza słów) – WordRepository.java, WordsApplication.java

```
public interface WordRepository extends  
JpaRepository<WordEntity, Integer> {  
}  
  
@SpringBootApplication  
public class WordsApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(  
            WordsApplication.class, args);  
    }  
}
```

# Prosta usługa REST (baza słów) – WordsController.java

```
@CrossOrigin  
@RestController  
public class WordsController {  
    @Autowired  
    WordRepository wordRepository;  
  
    @GetMapping("/api/words")  
    public List<WordEntity> getWords() {  
        List<WordEntity> words = wordRepository.findAll();  
        System.out.println("getWords() - words: " + words);  
        return words;  
    }  
  
    @PostMapping("/api/save_word")  
    public WordEntity saveWord(@Valid @RequestBody  
                               WordEntity newWord) {  
        WordEntity savedWord = wordRepository.save(newWord);  
        System.out.println("saveWord() - saved word: " + savedWord);  
        return savedWord;  
    }  
}
```

# Prosta usługa REST (baza słów) – WordsController.java

```
@GetMapping("/api/word/{id}")
public WordEntity getWord(@PathVariable(value = "id",
required = true) int wordId) throws Exception {
    WordEntity word =
        wordRepository.findById(wordId).orElseThrow(
            () -> new Exception(
                String.format("Word with %d id not found", wordId)));
    System.out.println("getWord() - word: " + word);
    return word;
}

@GetMapping("/api/delete_word/{id}")
public void deleteWord(@PathVariable(value = "id",
required = true) int wordId) {
    wordRepository.deleteById(wordId);
    System.out.println("deleteWord() - id: " + wordId);
}
```

# Prosta usługa REST (baza słów) – Swagger

<http://localhost:8080/swagger-ui.html>

The screenshot shows the Swagger UI interface for a 'words API' version 0.0.1. The left sidebar lists endpoints:

- DELETE /api/delete\_word/{id}** deleteWord
- POST /api/save\_word** saveWord
- GET /api/word/{id}** getWord
- GET /api/words** getWords

The right panel details the **GET /api/words getWords** endpoint:

- Parameters:** No parameters.
- Responses:** Response content type: \*/\*. The curl command is shown: `curl -X GET "http://localhost:8080/api/words" -H "accept: */*"`.
- Request URL:** `http://localhost:8080/api/words`
- Server response:** Status code 200, Response body: 

```
[{"id": 1, "word": "Spring"}]
```

# Klient usługi REST – Retrofit 2

- *Retrofit 2* to klient *REST* dla *Javy* i *Androida*
- Umożliwia on łatwe pobieranie i wysyłanie ustrukturyzowanych danych (np. w formacie *JSON*) za pośrednictwem usługi *REST*
- Możliwe jest dodanie/konfiguracja konwertera używanego do serializacji danych. W przypadku formatu *JSON* zazwyczaj używany jest *Gson*
- Do realizacji żądań http używana jest biblioteka *OkHttp*
- Użycie *Retrofit* wymaga zdefiniowania:
  - *klas modelu* - określają model *JSON*
  - *interfejsów* - określających dostępne operacje
  - klasa wykorzystującej *Retrofit.Builder* i interfejsu *Builder API*, umożliwiających zbudowanie klienta usługi o określonej konfiguracji

# Klient usługi REST – konfiguracja

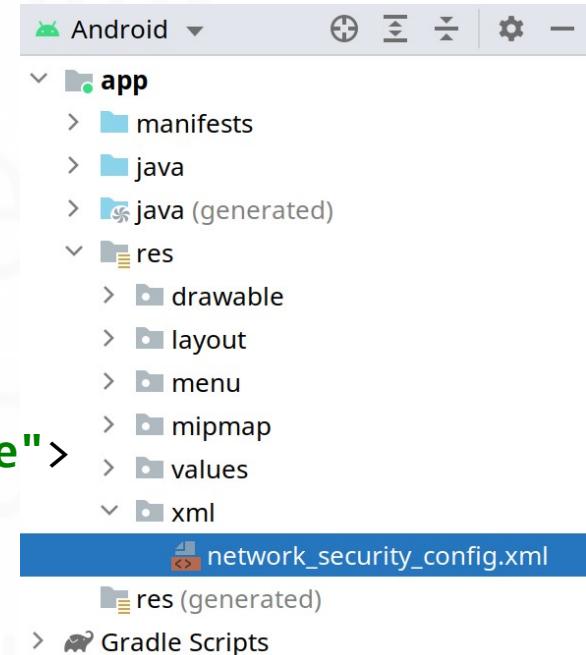
- Użycie połączeń sieciowych wymaga modyfikacji manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.w7_network_and">

    <uses-permission android:name="android.permission.INTERNET" />
    <!-- ... -->
</manifest>
```

- Nowe wersje Androida wymagają stosowania połączeń *https*. W przypadku połączeń neszzyfrowanych wymagane jest dodanie wyjątku

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">
            10.0.2.2</domain>
        </domain-config>
    </network-security-config>
```



# Klient usługi REST – zależności

- Biblioteka Retrofit 2 musi być dodana do zależności – plik build.gradle (moduł)

```
//...
dependencies {

    //...
    //Retrofit
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
    //...
}
```

- Przykład na kolejnych slajdach to modyfikacja bazy słów z wykładu 5-go. Klasy odpowiedzialne za wykorzystanie biblioteki Room (encja, DAO, repozytorium, baza danych...) zostaną zastąpione klasami korzystającymi z usługi REST
- Kod odpowiedzialny za interfejs użytkownika jest identyczny z poprzednim przykładem

# Klient usługi REST – encja

```
public class Word {  
    //Long aby zachować kompatybilność z GUI (w usłudze jest int)  
    public Long id;  
    public String word;  
  
    public Word(String word) {  
        this.word=word;  
    }  
  
    //gettery aby zachować kompatybilność z GUI  
    public String getWord() {  
        return word;  
    }  
  
    public Long getId() {  
        return id;  
    }  
}
```

# Klient usługi REST – definicja API usługi

```
public interface WordsApi {  
    String BASE_URL = "http://10.0.2.2:8080";  
  
    @Headers({"Accept:application/json",  
              "Content-Type:application/json"})  
    @GET("/api/words")  
    Call<List<Word>> getWords();  
  
    @Headers({"Accept:application/json",  
              "Content-Type:application/json"})  
    @POST("/api/save_word")  
    Call<Word> saveWord(@Body Word word);  
  
    @Headers({"Accept:application/json",  
              "Content-Type:application/json"})  
    @GET("/api/word/{id}")  
    Call<Word> getWord(@Path("id") Long id);  
  
    @Headers({"Accept:application/json",  
              "Content-Type:application/json"})  
    @DELETE("/api/delete_word/{id}")  
    Call<Void> deleteWord(@Path("id") Long id);  
}
```

# Klient usługi REST – tworzenie klienta

```
public class RetrofitSingleton {  
    private static Retrofit sTabletApiClient = null;  
  
    public static Retrofit getTabletApiClient() {  
        if (sTabletApiClient == null) {  
            sTabletApiClient = new Retrofit.Builder()  
                .baseUrl(WordsApi.BASE_URL)  
                .addConverterFactory(  
                    GsonConverterFactory.create())  
                .build();  
        }  
        return sTabletApiClient;  
    }  
}
```

# Klient usługi REST – repozytorium, konfiguracja

```
public class WordRepository {  
    private final static String TAG =  
        WordRepository.class.getSimpleName();  
  
    private MutableLiveData<List<Word>> mAllWords;  
    private Retrofit mRetrofit;  
    private WordsApi mWordsApi;  
  
    //parametr zachowany tylko dla kompatybilności z poprzednim  
    //przykładem  
    WordRepository(Application application) {  
        mAllWords = new MutableLiveData<>(new ArrayList<>());  
        mRetrofit = RetrofitSingleton.getTabletApiClient();  
        mWordsApi = mRetrofit.create(WordsApi.class);  
        getAllWordsFromService();  
    }  
  
    LiveData<List<Word>> getAllWords() {  
        return mAllWords;  
    }  
}
```

# Klient usługi REST – repozytorium, pobieranie danych

```
private void getAllWordsFromService() {
    mWordsApi.getWords().enqueue(new Callback<List<Word>>() {
        @Override
        public void onResponse(Call<List<Word>> call,
            Response<List<Word>> response) {
            Log.d(TAG, "getWords(), success");
            //postValue - bo z innego wątku
            mAllWords.postValue(response.body());
        }
        @Override
        public void onFailure(Call<List<Word>> call,
            Throwable t) {
            Log.d(TAG, "getWords(), error: " + t.getMessage());
        }
    });
}
```

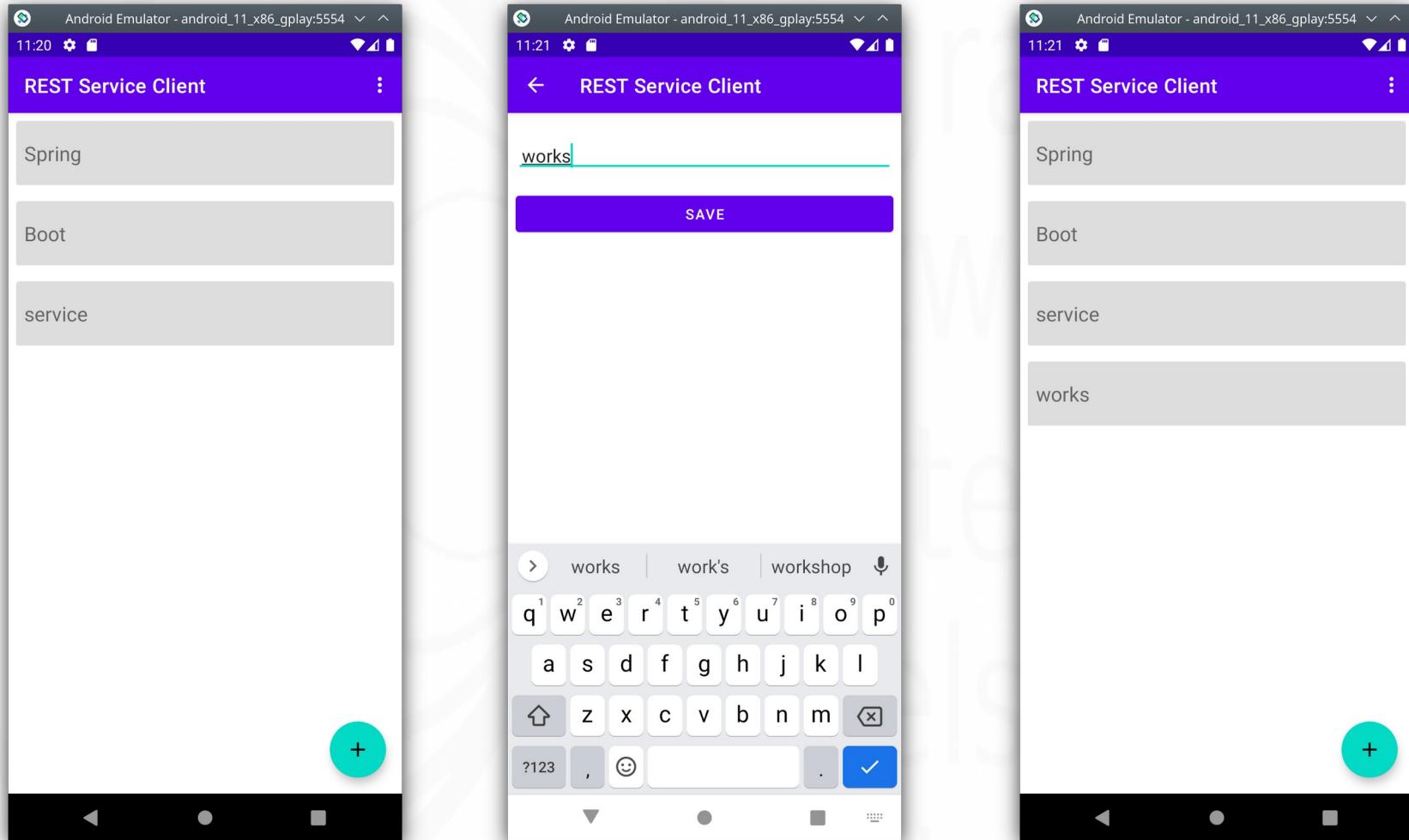
# Klient usługi REST – repozytorium, zapisywanie danych

```
void insert(Word word) {  
    mWordsApi.saveWord(word).enqueue(new Callback<Word>() {  
        @Override  
        public void onResponse(Call<Word> call,  
                               Response<Word> response) {  
            Log.d(TAG, "saveWord(), success");  
            getAllWordsFromService();  
        }  
  
        @Override  
        public void onFailure(Call<Word> call, Throwable t) {  
            Log.d(TAG, "saveWord(), error: " + t.getMessage());  
        }  
    });  
}
```

# Klient usługi REST – repozytorium, kasowanie danych

```
void deleteAll() {  
    //w API nie ma kasowania wszystkich :(  
    for (Word word : mAllWords.getValue()) {  
        deleteWord(word);  
    }  
}  
void deleteWord(Word word) {  
    mWordsApi.deleteWord(word.id).enqueue(new Callback<Void>() {  
        @Override  
        public void onResponse(Call<Void> call,  
                               Response<Void> response) {  
            Log.d(TAG, "deleteWord(), success");  
            getAllWordsFromService();  
        }  
  
        @Override  
        public void onFailure(Call<Void> call, Throwable t) {  
            Log.d(TAG, "deleteWord(), error: " + t.getMessage());  
        }  
    });  
}
```

# Klient usługi REST





# Programowanie Aplikacji Mobilnych na Platformę Android

## Wykład 8 – Wykorzystanie usług Google Play.

Jakub Smołka



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



# Usługi Google Play

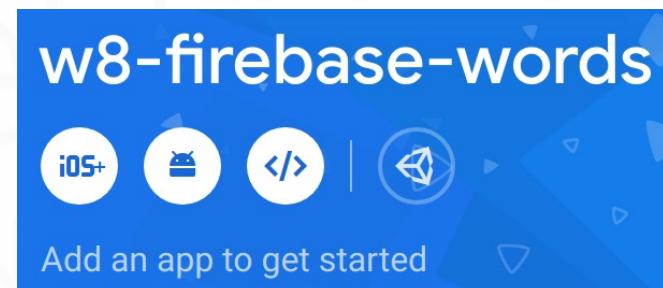
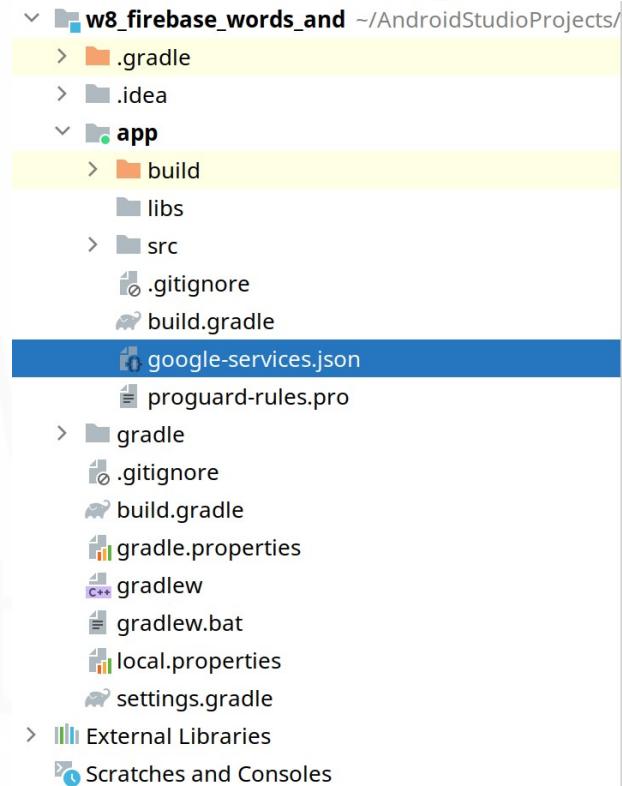
- *Usługi Google Play* udostępniają szereg API i usług w aplikacjach dla systemu Android. W ramach usług Google Play dostępne są:
  - *AdMob* - platforma reklam
  - *Analytics* - analiza danych nt. wykorzystania aplikacji
  - *Cast* - strumieniowanie dźwięku i wideo
  - *Fit* - śledzenie aktywności fizycznej
  - ***Firebase* - baza danych, uwierzytelnianie i szereg innych**
  - *Google Pay API* - integracja płatności w aplikacji
  - *Google Sign-In* - logowanie do aplikacji i autoryzowanie aplikacji do korzystania z usług Google
  - *Location and Context* - łączenie danych z wielu sensorów (rozpoznawanie aktywności, lokalizacja, geofencing...)
  - *Maps SDK* - mapy Google
  - *ML Kit* - uczenie maszynowe
  - *Nearby* - wykrywanie pobliskich urządzeń, przesyłanie danych
  - *Play Games Services* - usługi upraszczające tworzenie gier

# Firebase – przykład SDK w Google Play

- 2011 – *James Tamplin* i *Andrew Lee* założyli firmę *Firebase*. Pierwszy produkt – Realtime Database (API, które synchronizuje dane aplikacji na różnych urządzeniach i przechowuje je w chmurze).
- 2014 – dwa nowe produkty: Firebase Hosting i Firebase Authentication
- 2014 – Firebase zostało przejęte przez Google.
- 2016 – Firebase wprowadziło Firebase Analytics i ogłosiło, że rozszerza swoje usługi, aby stać się zunifikowaną platformą backend-as-a-service (BaaS) dla programistów aplikacji mobilnych.
- 2017 – Google nabył Fabric i Crashlytics od Twittera, aby dodać te usługi do Firebase
- 2017 – Firebase uruchomiło Cloud Firestore - dokumentową bazę danych czasu rzeczywistego (następca Realtime Database).

# Firebase – rejestracja projektu

- Wykorzystanie usług *Firebase* wymaga rejestracji projektu
- Na stronie (po zalogowaniu kontem Google):  
<https://console.firebaseio.google.com/>
  - wybrać *Add Project*
  - nadać nazwę projektowi
  - wyłączyć *Google Analytics* (nie będzie potrzebne w omawianych przykładach)
  - po utworzeniu projektu dodać aplikację (np. Android) – wymagane jest podanie nazwy pakietu aplikacji
  - pobrać plik `google-services.json` i dodać do projektu aplikacji mobilnej



# Firebase – zależności – build.gradle

- build.gradle (projekt)

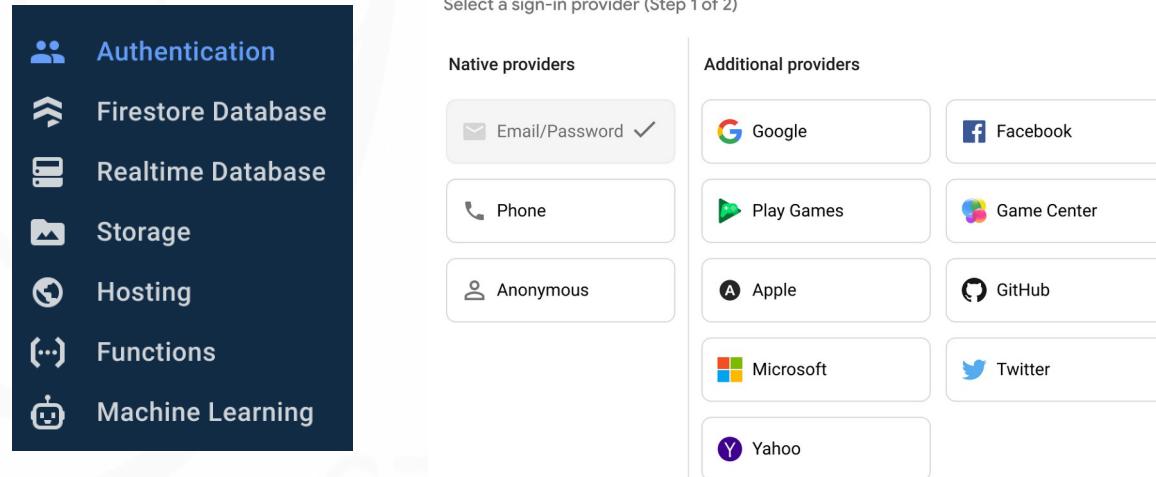
```
//...  
dependencies {  
    //...  
    //usługi Google Play  
    classpath 'com.google.gms:google-services:4.3.10'  
}  
//...
```

- build.gradle (moduł)

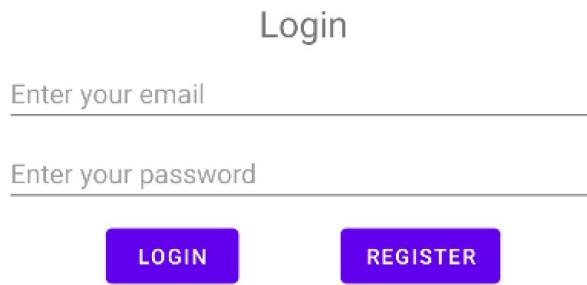
```
//...  
//usługi google  
apply plugin: 'com.google.gms.google-services'  
//...  
dependencies {  
    //...  
    //Firebase Bill of Materials  
    implementation platform('com.google.firebaseio.firebaseio-bom:29.0.3')  
    //potrzebne biblioteki firebase  
    //nie określamy wersji - dzięki dołączeniu BoM  
    //zawsze będą używane odpowiednie wersje  
    implementation 'com.google.firebaseio.firebaseio-auth'  
    implementation 'com.google.firebaseio.firebaseio-firestore'  
    //...  
}
```

# Fireauth – uwierzytelnianie użytkownika

- *Firebase Authentication* zapewnia usługi backendu oraz SDK do uwierzytelniania użytkowników w aplikacjach
- Obsługuje uwierzytelnianie za pomocą: haseł, numerów telefonów oraz popularnych federacyjnych dostawców tożsamości takich jak *Google*, *Facebook*, *Twitter*...
- Wybrany sposób uwierzytelniania trzeba uaktywnić w *konsoli Firebase*



# Fireauth – aktywność logowania – layout

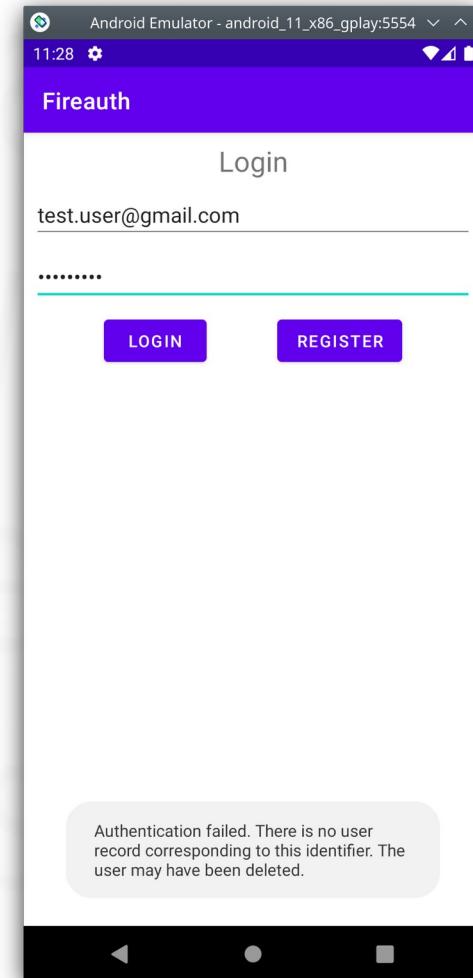
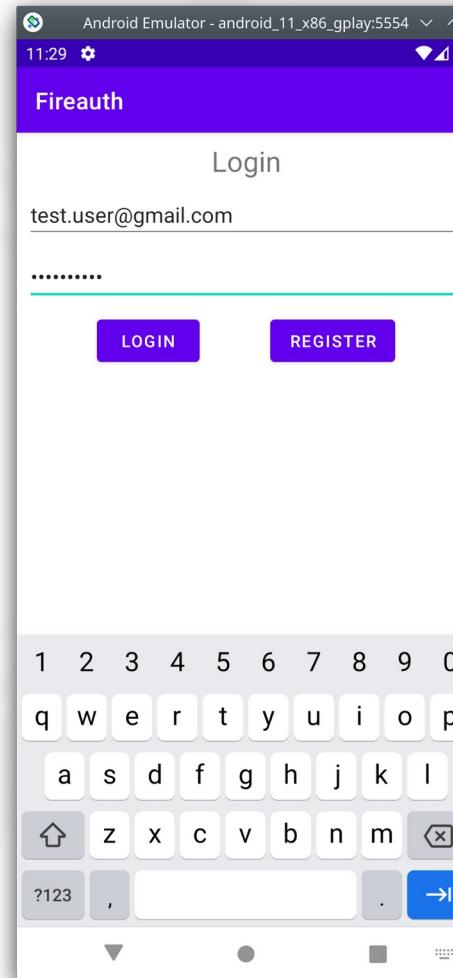
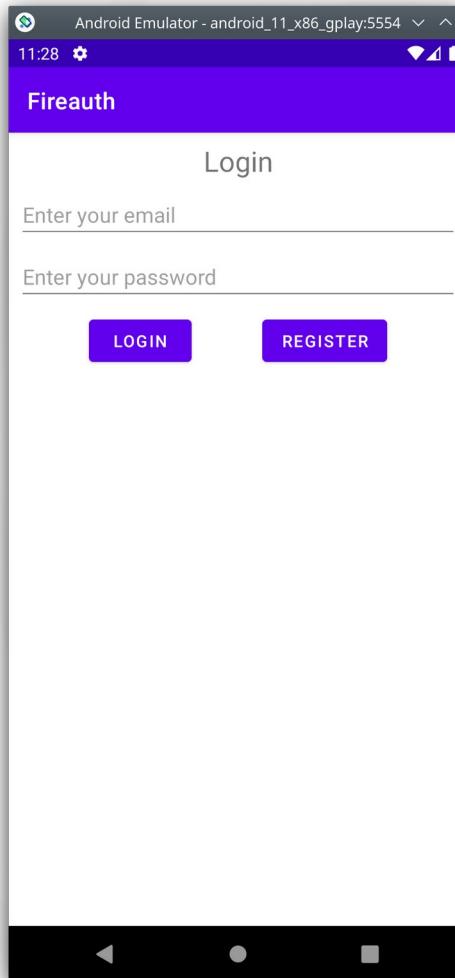


- Aktywność logowania jest prosta zawiera cztery istotne elementy:
  - activity\_login.xml
  - emailEditText
  - passwordEditText
  - loginButton
  - registerButton

# Fireauth – aktywność logowania

```
public class LoginActivity extends AppCompatActivity {  
    //onCreate() ustawia tylko Layout i obsługę kliknięcia przycisków...  
  
    private void login() {  
        EditText emailEditText = findViewById(R.id.emailEditText);  
        EditText passwordEditText = findViewById(R.id.passwordEditText);  
        String email = emailEditText.getText().toString();  
        String password = passwordEditText.getText().toString();  
  
        FirebaseAuth.getInstance()  
            .signInWithEmailAndPassword(email, password)  
            .addOnCompleteListener(this, task -> {  
                if (task.isSuccessful()) {  
                    Toast.makeText(LoginActivity.this,  
                        "Authentication success.",  
                        Toast.LENGTH_SHORT).show();  
                    Intent intent = new Intent(  
                        LoginActivity.this, MainActivity.class);  
                    startActivity(intent);  
                } else {  
                    Toast.makeText(LoginActivity.this,  
                        "Authentication failed. " +  
                        task.getException().getMessage(),  
                        Toast.LENGTH_SHORT).show();  
                }  
            });  
    }  
}
```

# Fireauth – uwierzytelnianie użytkownika



# Fireauth – aktywność rejestracji, layout

Register

Enter your email

Enter your password

Repeat password

**REGISTER**

- Aktywność rejestracji jest prosta zawiera cztery istotne elementy:
  - `activity_login.xml`
  - `emailEditText`
  - `passwordEditText`
  - `passwordRepeatEditText`
  - `registerButton`

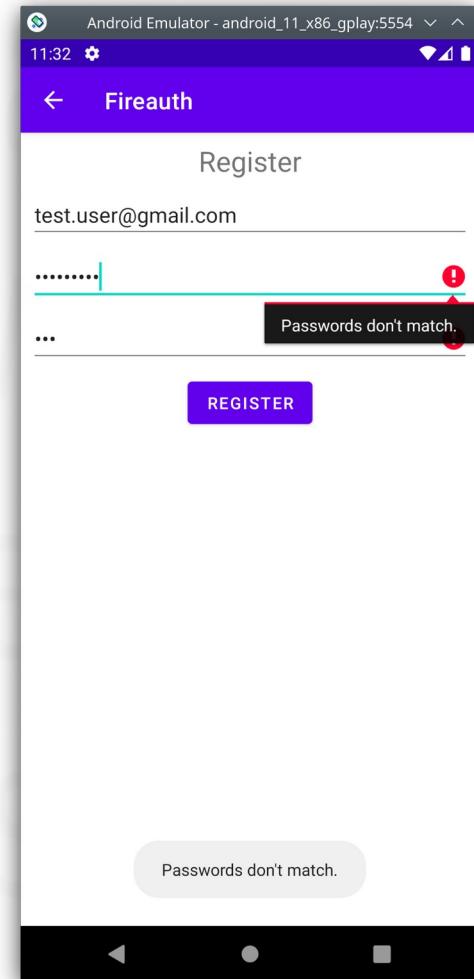
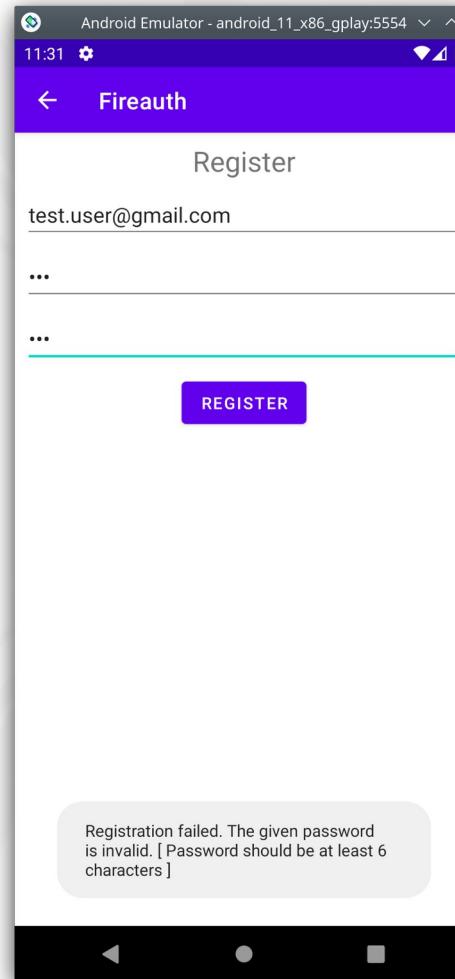
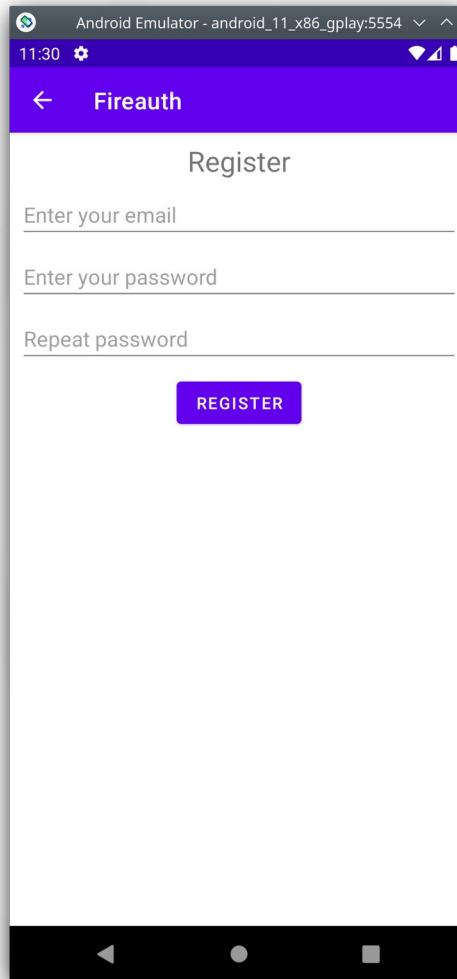
# Fireauth – aktywność rejestracji

```
public class RegisterActivity extends AppCompatActivity {  
    private static String TAG = LoginActivity.class.getSimpleName();  
  
    //onCreate() ustawia tylko Layout i obsługę kliknięcia przycisku...  
    private void register() {  
        EditText emailEditText = findViewById(R.id.emailEditText);  
        EditText passwordEditText =  
            findViewById(R.id.passwordEditText);  
        EditText passwordRepeatEditText =  
            findViewById(R.id.passwordRepeatEditText);  
        String email = emailEditText.getText().toString();  
        String password = passwordEditText.getText().toString();  
        String passwordRepeat =  
            passwordRepeatEditText.getText().toString();  
  
        if (!password.equals(passwordRepeat)) {  
            Toast.makeText(RegisterActivity.this,  
                "Passwords don't match.", Toast.LENGTH_SHORT)  
                .show();  
            passwordEditText.setError("Passwords don't match.");  
            passwordRepeatEditText.setError(  
                "Passwords don't match.");  
        } else {  
    }
```

# Fireauth – aktywność rejestracji

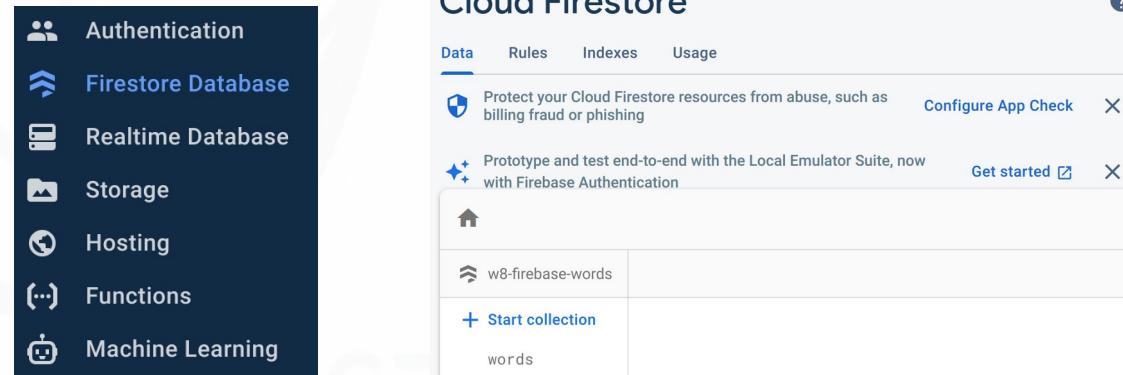
```
    FirebaseAuth.getInstance()
        .createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener(this, task -> {
            if (task.isSuccessful()) {
                Toast.makeText(RegisterActivity.this,
                    "Registration successful.",
                    Toast.LENGTH_SHORT).show();
                finish();
            } else {
                Toast.makeText(RegisterActivity.this,
                    "Registration failed. " +
                    task.getException().getMessage(),
                    Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

# Fireauth – rejestracja użytkownika



# Firestore – chmurowa baza NoSQL

- *Firebase* to elastyczna, skalowalna baza danych NoSQL w chmurze do przechowywania i synchronizowania danych
- Utrzymuje synchronizację danych między aplikacjami klienckimi za pośrednictwem odbiorników w czasie rzeczywistym
- Potrafi działać bez połączenia z Internetem, dzięki czemu można tworzyć responsywne aplikacje działające niezależnie od opóźnień w sieci lub łączności z Internetem



# Firestore – encja

- Przykład na kolejnych slajdach to modyfikacja bazy słów z wykładu 5-go/7-go. Klasy odpowiedzialne za pobieranie danych (encja, DAO, repozytorium...) zostaną zastąpione klasami korzystającymi z Firebase
- Kod odpowiedzialny za interfejs użytkownika jest identyczny z poprzednimi przykładami

```
public class Word {  
    public final static String KEY_USER_ID = "userId";  
    public final static String KEY_WORD = "word";  
  
    //dla zachowania kompatybilności z przykładem klienta REST  
    public Long id;  
    public String docId;  
    public String word;  
    public String userId;
```

# Firestore – encja

```
//konstruktor do tworzenia słów do dodania
public Word(String word) {
    //nowe słowa będą miały automatycznie ustawiane id
    //zalogowanego użytkownika
    userId = FirebaseAuth.getInstance().getUid();
    this.word = word;
}
//konstruktor do tworzenia słów pobranych z bazy
public Word(Long id, String docId, Map<String, Object> data) {
    this.id = id;
    this.docId = docId;
    this.word = (String) data.get(KEY_WORD);
    this.userId = (String) data.get(KEY_USER_ID);
}
//dane dokumentu określone są przez mapę
public Map<String, Object> toMap() {
    Map<String, Object> map = new HashMap<>();
    map.put(KEY_USER_ID, userId);
    map.put(KEY_WORD, word);
    return map;
}
//dla zachowania kompatybilności z wcześniejszym przykładem
public String getWord() { return word; }
public Long getId() { return id; }
}
```

# Firestore – repozytorium definicje

```
public class WordRepository {  
    private final static String TAG =  
        WordRepository.class.getSimpleName();  
  
    private final static String COLLECTION = "words";  
  
    private MutableLiveData<List<Word>> mAllWords;  
  
    //parametr zachowany tylko dla kompatybilności z wcześniejszym  
    //przykładem  
    WordRepository(Application application) {  
        mAllWords = new MutableLiveData<>(new ArrayList<>());  
        getAllWordsFromFirebase();  
    }  
  
    //id będzie potrzebne do zachowania kompatybilności z przykładem  
    //klienta REST (adapter, selectiontracker...)  
    long mLongId = 0L;
```

# Firestore – repozytorium, pobieranie danych, konwersja dokumentów na listę słów

```
private void getAllWordsFromFirebase() {
    mLongId = 0L;
    FirebaseFirestore.getInstance()
        .collection(COLLECTION)
        .whereEqualTo(Word.KEY_USER_ID,
            FirebaseAuth.getInstance().getUid())
        .get().addOnCompleteListener(task -> {
    if (task.isSuccessful()) {
        List<Word> wordList = new ArrayList<>();
        //wynikiem zadania jest migawka zapytania
        //przeglądamy migawki poszczególnych dokumentów
        for (QueryDocumentSnapshot document : task.getResult()) {
            Word word = new Word(++mLongId,
                document.getId(), document.getData());
            wordList.add(word);
        }
        mAllWords.setValue(wordList);
    } else {
        Log.d(TAG, "getAllWordsFromService()" +
            "addOnCompleteListener " +
            "- Error getting documents: ",
            task.getException());
    }
});}
```

# Firestore – repozytorium, zapisywanie danych

```
LiveData<List<Word>> getAllWords() {  
    return mAllWords;  
}  
  
void insert(Word word) {  
    FirebaseFirestore.getInstance()  
        .collection(COLLECTION)  
        .document().set(word.toMap())  
        .addOnCompleteListener(task -> {  
            if (!task.isSuccessful()) {  
                Log.d(TAG, "insert()/addOnCompleteListener" +  
                    " - Error inserting document: ",  
                    task.getException());  
            }  
            getAllWordsFromFirebase();  
        });  
}
```

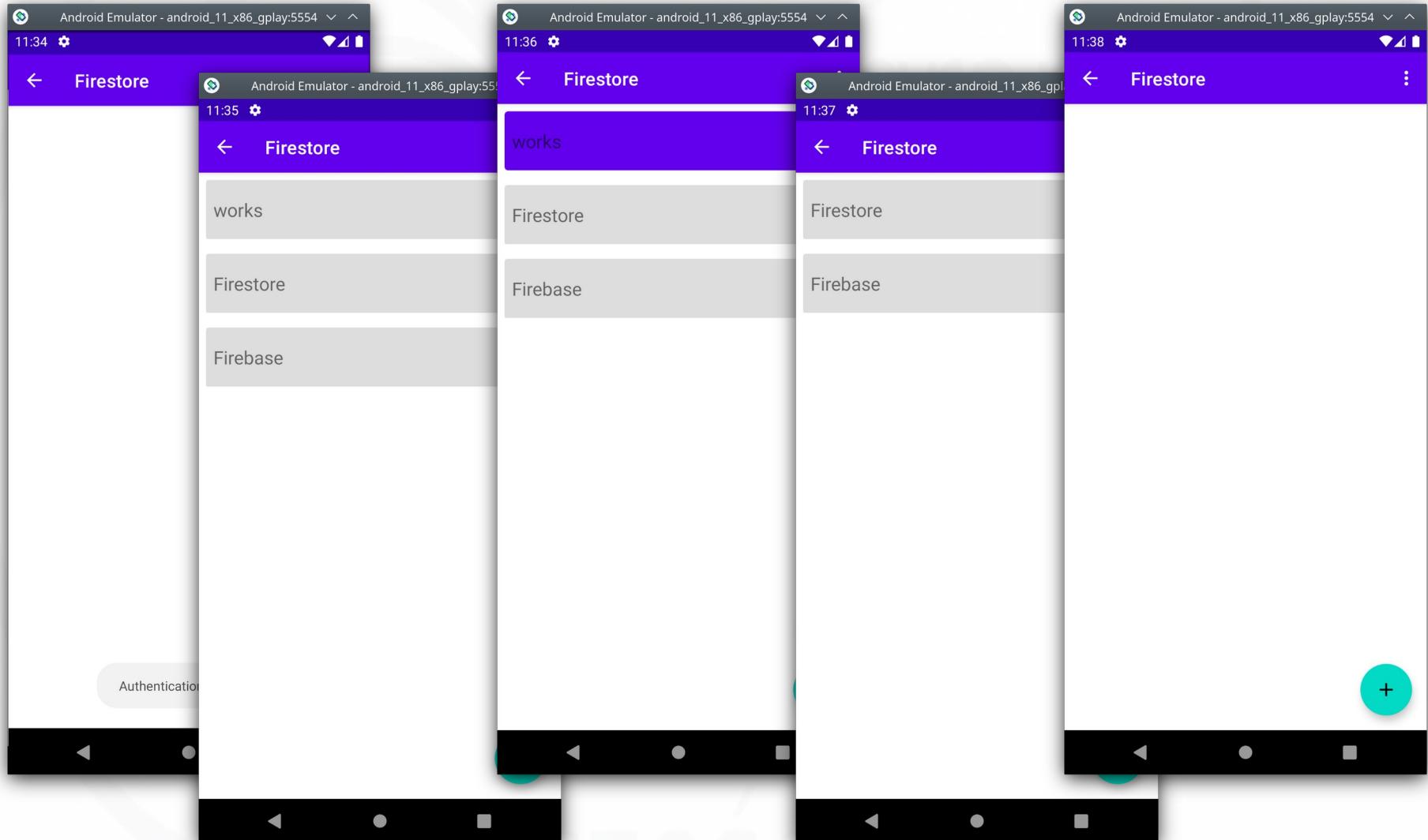
# Firestore – repozytorium, usuwanie danych, przetwarzanie wsadowe

```
void deleteAll() {
    FirebaseFirestore firebaseFirestoreInstance =
        FirebaseFirestore.getInstance();
    //umożliwia przetwarzanie wsadowe (metody delete(), set(),
    //update())
    WriteBatch writeBatch = firebaseFirestoreInstance.batch();
    firebaseFirestoreInstance.collection(COLLECTION)
        .whereEqualTo(Word.KEY_USER_ID,
            FirebaseAuth.getInstance().getUid())
        .get().addOnCompleteListener(task -> {
    if (task.isSuccessful()) {
        for (QueryDocumentSnapshot document :
            task.getResult()) {
            writeBatch.delete(document.getReference());
        }
    }
    //zatwierdzamy kasowanie wszystkich
    writeBatch.commit();
    getAllWordsFromFirebase();
});}
```

# Firestore – repozytorium, usuwanie danych

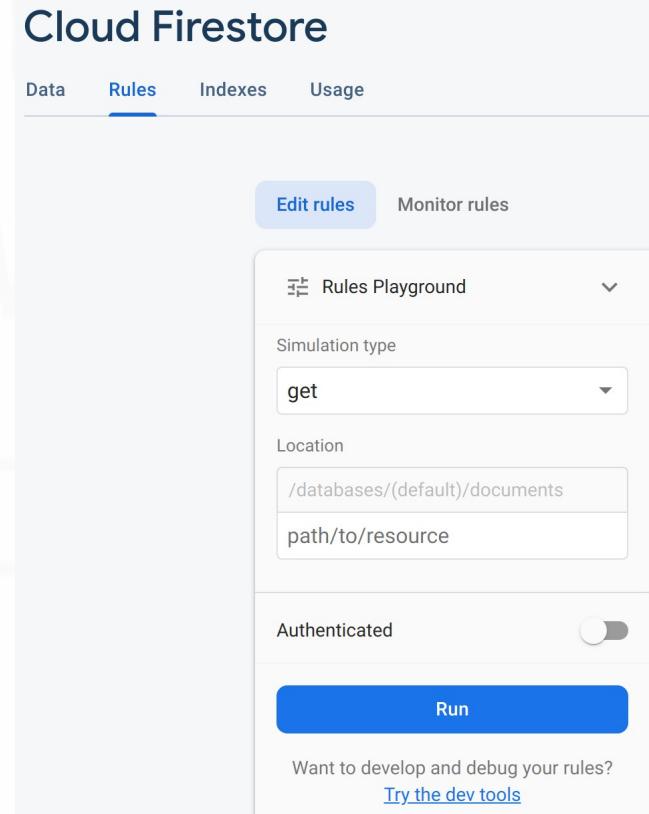
```
void deleteWord(Word word) {
    FirebaseFirestore.getInstance()
        .collection(COLLECTION)
        .document(word.docId)
        .delete().addOnCompleteListener(task -> {
            if (!task.isSuccessful()) {
                Log.d(TAG, "deleteWord()/addOnCompleteListener" +
                    " - Error deleting document: ",
                    task.getException());
            }
        });
    getAllWordsFromFirebase();
}
```

# Firestore



# Firestore – reguły dostępu

- *Cloud Firestore* może zostać utworzone z zablokowanym dostępem lub w trybie debugowania z pełnym dostępem przez miesiąc od utworzenia projektu
- Warto zmodyfikować zestaw reguł aby lepiej kontrolować dostęp użytkowników do danych (i zablokować dostęp osobom postronnym)
- *Konsola Firestore* zawiera:
  - edytor reguł
  - symulator żądań
  - monitor reguł



# Firestore – reguły dostępu

```
//wersja języka zasad
rules_version = '2';
//nazwa usługi do której zasady mają zastosowanie
service cloud.firestore {
    //ścieżka dopasowania, {} - wildcards -> wartość ze ścieżki
    //zostanie wstawiona do zmiennej np database
    match /databases/{database}/documents {
        /** - dopasowanie rekursywne (dowolnego dokumentu nawet w
        //zagnieżdżonej kolekcji)
        match /words/{document=**} {
            function isLoggedIn() {
                return request.auth.uid !=null;
            }
            function isNewOwner() {
                //request.auth - informacje o użytkowniku w żądaniu
                //request.resource - dokument do zapisania/zmodyfikowania
                return request.auth.uid == request.resource.data.userId;
            }
            function isCurrentOwner() {
                //resource - odczytywany/modyfikowany/kasowany dokument
                //aktualnie w bazie
                return resource.data.userId == request.auth.uid;
            }
        }
    }
}
```

# Firebase – reguły dostępu

```
function isCorrectData() {  
    return request.resource.data.keys()  
        .hasAll(["word", "userId"]);  
}  
  
allow get, list: if isLoggedIn() && isCurrentOwner();  
allow create: if isLoggedIn() && isCorrectData() &&  
    isNewOwner();  
allow delete: if isLoggedIn() && isCurrentOwner();  
allow update: if isLoggedIn() && isCurrentOwner();  
//do debugowania - zezwala na wszystko  
//allow read, write: if true;  
//do debugowania - zezwala na wszystko przez określony czas  
//allow read, write: if  
//    request.time < timestamp.date(2022, 1, 10);  
}  
}  
}
```



# Programowanie Aplikacji Mobilnych na Platformę Android

## Wykład 9 – Modyfikacja standardowego wyglądu komponentów graficznych.

Jakub Smołka



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



# Style / motyw

- *Styl* – zbiór atrybutów, które określają wygląd pojedynczego widoku
  - styl może określać atrybuty, które mogą być ustawione dla danego typu widoku np. kolor czcionki, rozmiar czcionki, czy tło
  - atrybuty nieobsługiwane przez widok są ignorowane
- *Motyw* - zbiór atrybutów, które są stosowane do całej aplikacji, aktywności lub widoków (nie tylko do pojedynczego widoku)
  - Po zastosowaniu motywu każdy widok w aplikacji lub aktywności stosuje wszystkie obsługiwane atrybuty motywu
  - Motywów mogą również stosować style do elementów innych niż widok np. pasek stanu i tło okna
  - Style / motyw definiuje się w katalogu /res/values zazwyczaj w pliku o nazwie **styles.xml** / **themes.xml**

# Tworzenie stylu

- Styl można utworzyć w następujący sposób:

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <style name="MyText">  
        <item name="android:textSize">10sp</item>  
        <item name="android:textColor">#00FF00</item>  
    </style>  
</resources>
```

- Styl można zastosować do widoku za pomocą atrybutu style:

```
<TextView  
    style="@style/MyText"  
/>
```

# Rozszerzanie stylu

- Styl można utworzyć na bazie innego poprzez użycie atrybutu `parent` np.:

```
<style name="MyText"  
parent="TextAppearance.AppCompat">  
    <item name="android:textSize">10sp</item>  
    <item name="android:textColor">#00FF00</item>  
</style>
```

- Rozszerzyć styl można używając zapisu z kropką bez atrybutu `parent` (nie dotyczy stylów platformy) np.:

```
<style name="MyText.Large">  
    <item name="android:textSize">20sp</item>  
</style>
```

# Dostępne atrybuty

- Dostępne widoki uwzględniają bardzo dużą liczbę atrybutów
- Atrybuty dostępne dla konkretnych widoków można znaleźć
  - w dokumentacji klasy widoku na stronie [developer.android.com](https://developer.android.com) w części *XML Attributes* lub *Inherited XML Attributes*
  - w dokumentacji *Material Design* na stronie <https://material.io/components?platform=android> (po wybraniu komponentu, zakładka *implementation*)
- Przy tworzeniu stylu / motywu pomocne może być narzędzie Color Tool (<https://material.io/resources/color/>). Pozwala ono szybko sprawdzić wygląd aplikacji z określonym zestawem kolorów oraz wybrać kolory pasujące do siebie

# Stosowanie stylu jako motywu

- Stworzony styl można zastosować do wybranego elementu aplikacji jako motyw w pliku manifestu używając atrybutu android:theme.

Motyw można zastosować do całej aplikacji

```
<manifest ... >
    <application android:theme="@style/Theme.MyApp.MyTheme" ... >
        </application>
    </manifest>
```

- lub do wybranej aktywności

```
<manifest ... >
    <application ... >
        <activity android:theme="@style/Theme.MyApp.MyTheme" ... >
            </activity>
        </application>
    </manifest>
```

- Motyw można również zastosować do konkretnego widoku w pliku layoutu (również za pomocą atrybutu android:theme). Motyw będzie zastosowany do widoku i widoków zagnieżdzonych

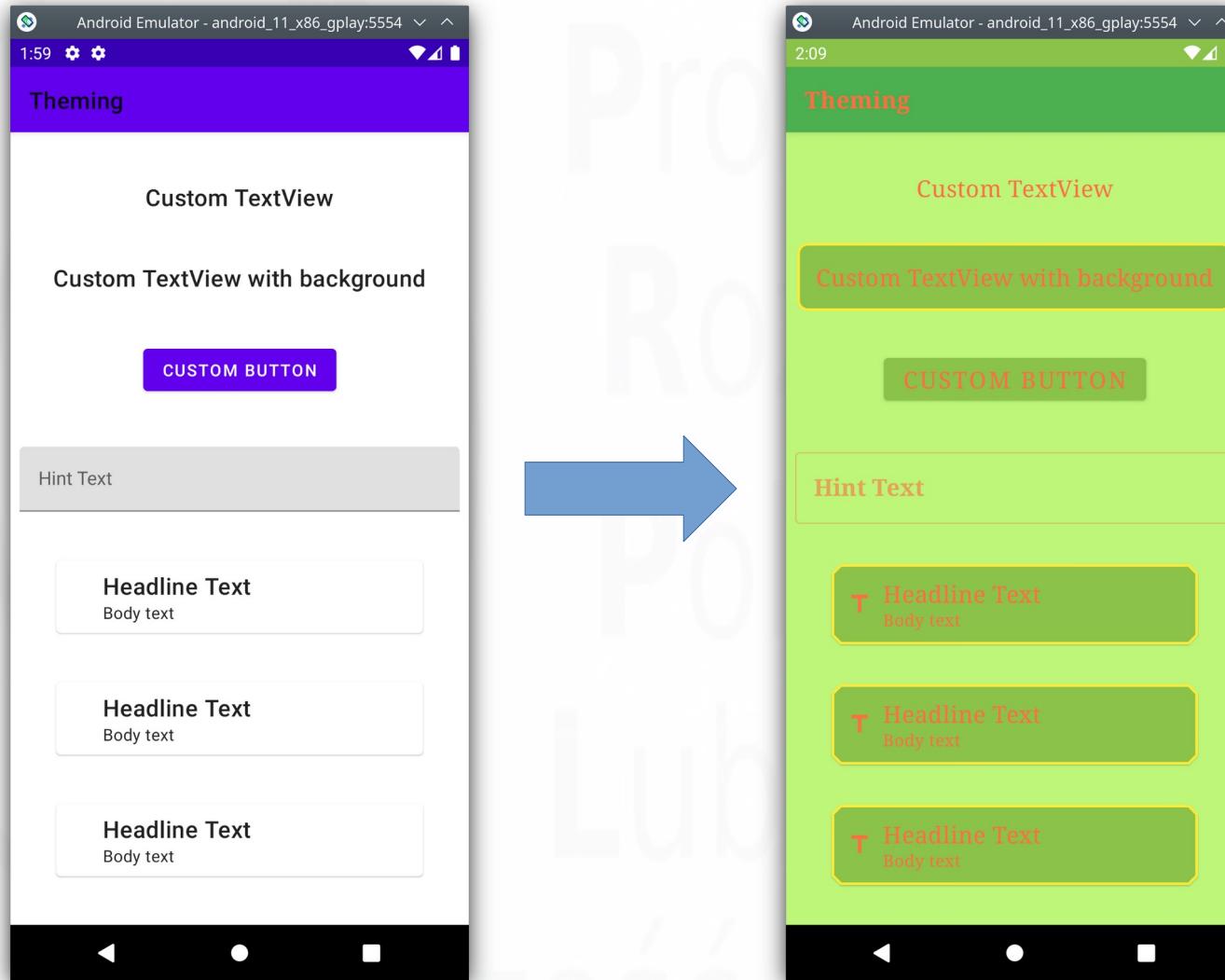
# Atrybuty w motywach

- Atrybuty, które można określić w motywach można znaleźć pod adresem:  
<https://developer.android.com/reference/android/R.styleable#Theme>
- W przypadku komponentów Material Design bardzo dobrym źródłem jest ponownie dokumentacja na stronie  
<https://material.io/components?platform=android>  
(po wybraniu komponentu, zakładka *implementation*)
  - znajdują się tam przykłady gotowych stylów dla wszystkich komponentów Material Design (również dla innych frameworków np. Flutter)

# Hierarchia stylów

- Atrybut może być określony w wielu miejscach
- Poniżej przedstawiono listę priorytetów (od najwyższego do najniższego)
  - style określone na poziomie znaków lub akapitów za pomocą zakresów tekstu (ang. spans) w komponentach pochodnych TextView
  - atrybuty określone programowo (w kodzie Java)
  - atrybuty zastosowane bezpośrednio do konkretnego widoku (w pliku layoutu)
  - style zastosowane do widoku
  - style domyślne
  - motyw zastosowane do kolekcji widoków, aktywności lub całej aplikacji
  - zastosowanie stylów specyficznych dla widoku, takich jak ustawienie TextAppearance w komponencie TextView

# Motyw



# Motywy – manifest aplikacji

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=
    "http://schemas.android.com/apk/res/android"
    package="com.example.w9_styles_and">

    <application
        <!-- ... -->
        android:theme="@style/Theme.W9_theming">
        <!-- ... -->
    </application>
</manifest>
```

# Motywy – layout, główna aktywność – pasek aplikacji

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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">

    <!-- aby ostylować pasek aplikacji trzeba go umieścić w Layoucie -->
    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/topAppBarLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <com.google.android.material.appbar.MaterialToolbar
            android:id="@+id/topAppBar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            app:title="@string/app_name" />

    </com.google.android.material.appbar.AppBarLayout>
```

# Motywy – layout, główna aktywność – przyciski

```
<TextView  
    android:id="@+id/textView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Custom TextView"  
    android:textAppearance="?attr/textAppearanceHeadline6"  
    app:layout_constraintBottom_toTopOf="@+id/textView5"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/topAppBarLayout" />
```

```
<TextView  
    android:id="@+id/textView5"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Custom TextView with background"  
    android:textAppearance="?attr/textAppearanceHeadline6"  
    app:layout_constraintBottom_toTopOf="@+id/button"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/textView" />
```

# Motywy – layout, główna aktywność – pole tekstowe

```
<com.google.android.material.textfield.TextInputLayout  
    android:id="@+id/textInputLayout"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_marginStart="8dp"  
    android:layout_marginEnd="8dp"  
    android:hint="Hint Text"  
    app:layout_constraintBottom_toTopOf="@+id/cardView"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/button">  
  
<com.google.android.material.textfield.TextInputEditText  
    android:id="@+id/textInputEditText"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" />  
  
</com.google.android.material.textfield.TextInputLayout>
```

# Motywy – layout, główna aktywność – karta

```
<com.google.android.material.card.MaterialCardView  
    android:id="@+id/cardView"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    app:layout_constraintBottom_toTopOf="@+id/cardView2"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/textInputLayout"  
    app:layout_constraintWidth_percent="0.8">  
  
<LinearLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:orientation="horizontal"  
    android:padding="8dp">  
  
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center_vertical"  
    app:srcCompat="@drawable/ic_baseline_title_24" />
```

# Motywy – layout, główna aktywność – karta

```
<LinearLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_marginStart="8dp"  
    android:orientation="vertical">  
    <TextView  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:text="Headline Text"  
        android:textAppearance=  
            "?attr/textAppearanceHeadline6" />  
    <TextView  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:text="Body text"  
        android:textAppearance=  
            "?attr/textAppearanceBody2" />  
    </LinearLayout>  
  </LinearLayout>  
</com.google.android.material.card.MaterialCardView>  
<!-- karta powtórzona jeszcze 2 razy ... --&gt;<br/></androidx.constraintlayout.widget.ConstraintLayout>
```

# Motywy – definiowanie kształtów

- W plikach XML można definiować podstawowe kształty: prostokąt, owal, linia, pierścień. Pliki zapisuje się w /res/drawable
- Można określić: rozmiar, kolor wypełnienia, szerokość i inne, właściwości linii, gradient wypełnienia, zaokrąglenie rogów, odstępy (padding)
- poniżej zdefiniowano zielony prostokąt z zaokrąglonymi rogami i żółtą ramką

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">

    <solid android:color="@color/light_green" />
    <stroke
        android:width="2dp"
        android:color="@color/yellow" />
    <corners android:radius="8dp" />

</shape>
```

# Motywy – definiowanie motywów aplikacji

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- pasek dodamy w Layoucie bo ma większe możliwości ostylowania -->
    <style name="Theme.W9_theming" parent=
        "Theme.MaterialComponents.DayNight.NoActionBar">

        <!-- pasek stanu - zmodyfikowane (było standardowo) -->
        <item name="android:statusBarColor"
            tools:targetApi="1">@color/light_green</item>

        <!-- dodatkowe ustawienia ... -->

        <!-- globalnie -->
        <item name="android:textColor">@color/deep_orange</item>
        <item name="fontFamily">serif</item>
        <item name="android:fontFamily">serif</item>

        <!-- tło aktywności -->
        <item name="android:windowBackground">
            @color/light_green_accent</item>

        <!-- pasek aplikacji -->
        <item name="toolbarStyle">@style/Widget.W9_Theming.MyToolbar</item>
```

# Motywy – definiowanie motywów aplikacji

```
<!-- pole tekstowe -->
<item name="textInputStyle">
    @style/Widget.W9_Theming.MyTextInputLayout</item>

<!-- karta -->
<item name="materialCardViewStyle">
    @style/Widget.W9_Theming.MyCardView</item>

<!-- przycisk -->
<item name="materialButtonStyle">
    @style/Widget.W9_Theming.MyButton</item>
</style>
```

Custom TextView

# Motywy – styl paska aplikacji

```
<!-- pasek aplikacji -->
<style name="Widget.W9_Theming.MyToolbar"
    parent="Widget.MaterialComponents.Toolbar.Primary">
    <item name="materialThemeOverlay">
        @style/ThemeOverlay.W9_Theming.MyToolbar</item>
        <item name="titleTextAppearance">
            @style/TextAppearance.W9_Theming.Headline6.MyToolbar</item>
            <item name="titleTextColor">@color/deep_orange</item>
    </style>

<!-- zastępuje atrybuty zdefiniowane globalnie -->
<style name="ThemeOverlay.W9_Theming.MyToolbar" parent="">
    <item name="colorPrimary">@color/green</item>
</style>

<style name="TextAppearance.W9_Theming.Headline6.MyToolbar"
    parent="TextAppearance.MaterialComponents.Headline6">
    <item name="fontFamily">serif</item>
    <item name="android:textSize">20sp</item>
    <item name="android:textStyle">bold</item>
</style>
```



Theming

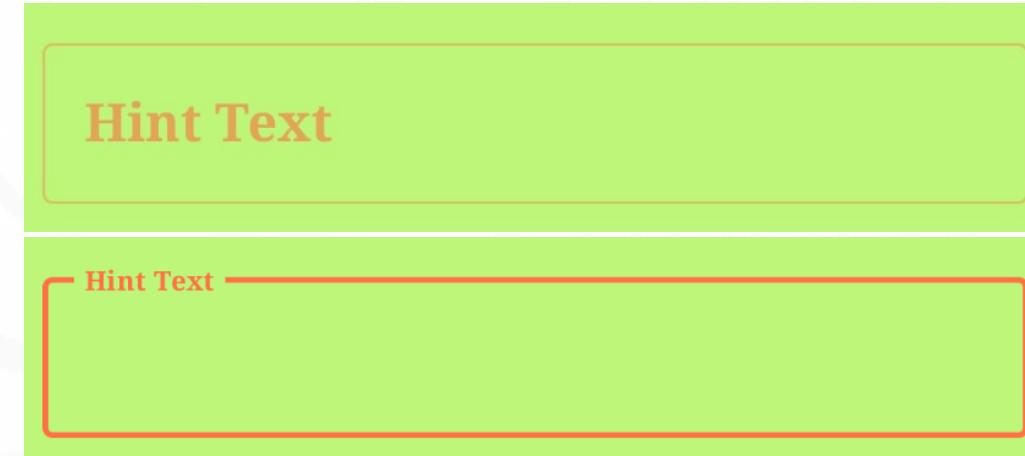
# Motyw – styl pola tekstowego

```
<!-- pole tekstowe -->
<style name="Widget.W9_Theming.MyTextInputLayout"
    parent="Widget.MaterialComponents.TextInputLayout.OutlinedBox">
    <item name="materialThemeOverlay">
        @style/ThemeOverlay.W9_Theming.MyTextInputLayout</item>
    </style>
<style name="ThemeOverlay.W9_Theming.MyTextInputLayout" parent="">
    <!-- koniecznie trzeba ustawić poniższy atrybut jeżeli używa
        się ThemeOverlay z TextInputLayout, w przeciwnym razie pole
        nie będzie miało właściwego stylu -->
    <item name="editTextStyle">
        @style/Widget.MaterialComponents.TextInputEditText.OutlinedBox
    </item>
    <item name="textAppearanceSubtitle1">
        @style/TextAppearance.W9_Theming.Subtitle1.MyTextInputLayout
    </item>
    <item name="textAppearanceCaption">
        @style/TextAppearance.W9_Theming.Caption.MyTextInputLayout
    </item>
    <item name="hintTextColor">@color/deep_orange</item>
    <item name="colorPrimary">@color/deep_orange</item>
    <item name="colorOnSurface">@color/deep_orange</item>
    <item name="colorError">@color/deep_orange</item>
</style>
```

# Motyw – styl pola tekstowego

```
<!-- tekst w polu -->
<style name="TextAppearance.W9_Theming.Subtitle1.MyTextInputLayout"
    parent="TextAppearance.MaterialComponents.Subtitle1">
    <item name="fontFamily">serif</item>
    <item name="android:textSize">20sp</item>
    <item name="android:textStyle">bold</item>
</style>

<!-- podpowiedź -->
<style name="TextAppearance.W9_Theming.Caption.MyTextInputLayout"
    parent="TextAppearance.MaterialComponents.Caption">
    <item name="fontFamily">serif</item>
    <item name="android:textSize">10sp</item>
    <item name="android:textStyle">bold</item>
</style>
```



# Motyw – styl karty

```
<!-- karta -->
<style name="Widget.W9_Theming.MyCardView"
    parent="Widget.MaterialComponents.CardView">
    <item name="materialThemeOverlay">
        @style/ThemeOverlay.W9_Theming.MyCardView</item>
    <item name="shapeAppearance">
        @style/ShapeAppearance.W9_Theming.MediumComponent.MyCardView
    </item>
    <item name="strokeColor">@color/yellow</item>
    <item name="strokeWidth">2dp</item>
    <item name="cardBackgroundColor">@color/light_green</item>
</style>

<style name="ThemeOverlay.W9_Theming.MyCardView" parent="">
    <item name="colorSurface">@color/light_green</item>
    <item name="android:tint">@color/deep_orange</item>
    <item name="android:textColor">@color/deep_orange</item>
    <item name="fontFamily">serif</item>
</style>
<style name="ShapeAppearance.W9_Theming.MediumComponent.MyCardView"
    parent="ShapeAppearance.MaterialComponents.MediumComponent">
    <item name="cornerFamily">cut</item>
    <item name="cornerSize">8dp</item>
</style>
```



# Motywy – styl przycisku

```
<!-- przycisk -->
<style name="Widget.W9_Theming.MyButton"
    parent="Widget.MaterialComponents.Button">
    <item name="materialThemeOverlay">
        @style/ThemeOverlay.W9_Theming.MyButton</item>
    <item name="android:textAppearance">
        @style/TextAppearance.W9_Theming.MyButton</item>
    <item name="android:textColor">@color/deep_orange</item>
    <item name="rippleColor">@color/light_blue_accent</item>
</style>
<style name="ThemeOverlay.W9_Theming.MyButton" parent="">
    <item name="colorPrimary">@color/light_green</item>
</style>
<style name="TextAppearance.W9_Theming.MyButton"
    parent="TextAppearance.MaterialComponents.Button">
    <item name="fontFamily">serif</item>
    <item name="android:fontFamily">serif</item>
    <item name="fontWeight">700</item>
    <item name="android:textSize">20sp</item>
</style>
</resources>
```

CUSTOM BUTTON

CUSTOM BUTTON

# Motywy – styl z tłem etykiety

- Styl definiujący ramkę elementu (themes.xml)

```
<!-- ramka, określamy parent ="" aby nie próbować dziedziczyć po  
Widget.W9_Theming -->  
<style name="Widget.W9_Theming.MyFrame" parent="">  
    <item name="android:padding">16dp</item>  
    <item name="android:background">  
        @drawable/light_green_rounded_box_yellow_border</item>  
</style>
```

- Zastosowanie stylu do elementu (modyfikacja layoutu)

```
<TextView  
    <!-- ... dodajemy styl -->  
    style="@style/Widget.W9_Theming.MyFrame"  
    android:text="Custom TextView with background"  
    <!-- ... -->  
/>
```

Custom TextView with background



# Programowanie Aplikacji Mobilnych na Platformę Android

## Wykład 10 – Wstęp do multimedów na platformie Android.

Jakub Smołka



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny

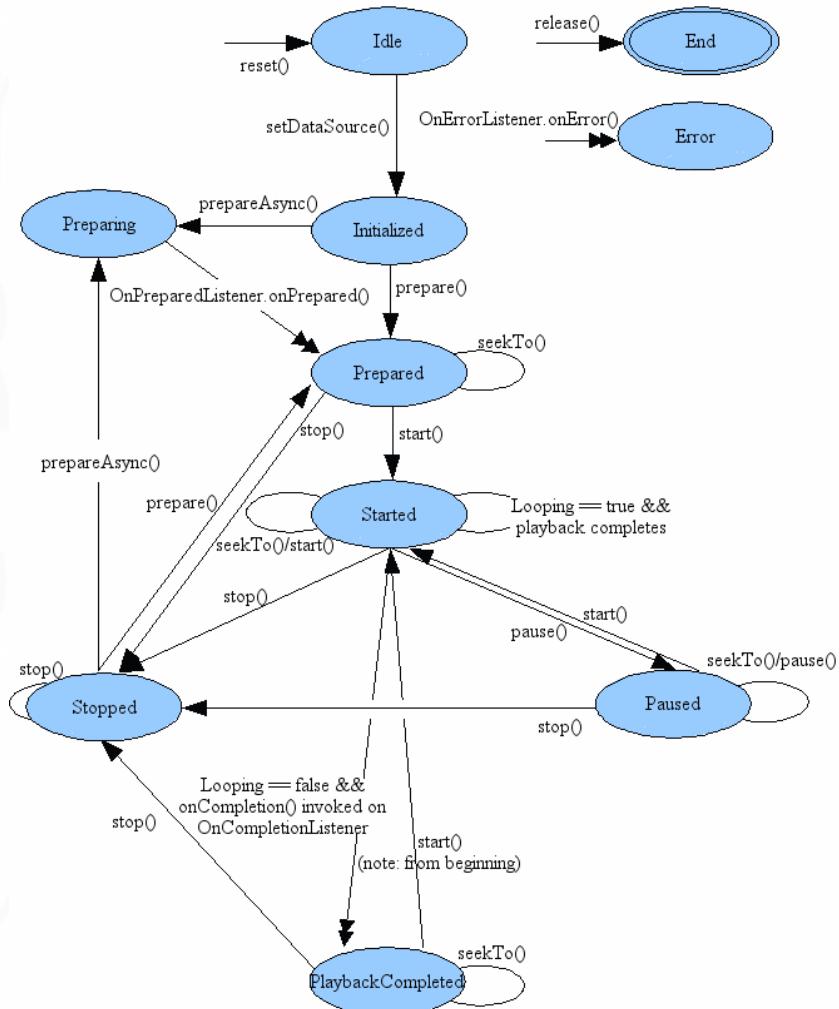


# Multimedia – klasa MediaPlayer

- Klasa MediaPlayer w systemie Android umożliwia odtwarzanie multimediiów z:
  - lokalnych zasobów
  - wewnętrznych źródeł danych (pliki, dostawcy treści – URI)
  - zewnętrznych źródeł określonych za pomocą adresów URL
- Użycie polega na:
  - utworzeniu obiektu MediaPlayer
  - ustawieniu źródła mediów
  - ustawieniu strumienia wyjściowego (opcjonalnie)
  - ustawieniu obsługi zdarzenia – OnPreparedListener – oznaczającego, że dostatecznie dużo danych zostało zbuforowanych
  - wywoływaniu metod: start(), stop(), pause(), seekTo()
  - zwolnieniu odtwarzacza za pomocą metody release()

# Diagram stanów MediaPlayer

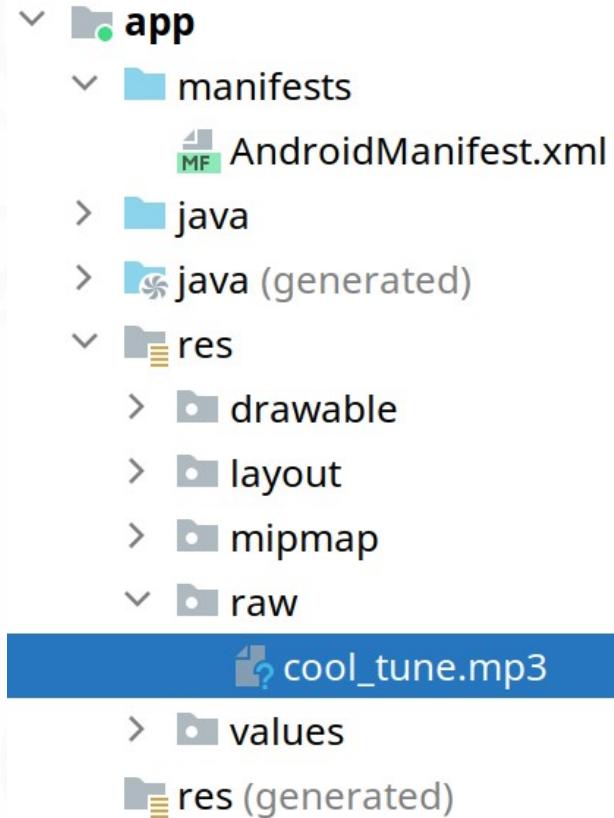
- Odtwarzacz może znajdować się w jednym wielu stanów (podobnie jak automat skończony)
- Ważne jest aby wywoływać tylko metody dozwolone w określonym stanie (inaczej wyjątek)



źródło: <https://developer.android.com/reference/android/media/MediaPlayer>

# Multimedia – zasoby

- Przykładowa aplikacja będzie wykorzystywać odtwarzacz MediaPlayer
- Aplikacja odtwarza nagranie z zasobów projektu
- Nagranie w formacie mp3 zostało dodane do katalogu res/raw



# Multimedia – główna aktywność, layout



- Aktywność odtwarzacza jest prosta zawiera siedem istotnych elementów:
  - `activity_main.xml`
  - `seekBar`
  - `currentTimeTextView`
  - `totalTimeTextView`
  - `rewindButton`
  - `fastForwardButton`
  - `pauseButton`
  - `playButton`

# Multimedia – główna aktywność, deklaracje

```
public class MainActivity extends AppCompatActivity {

    private static final String TAG =
        MainActivity.class.getSimpleName();

    // pola tekstowe i pasek przewijania...
    private TextView mCurrentTimeTextView;
    private TextView mTotalTimeTextView;
    private SeekBar mSeekBar;
    // przyciski sterujące potrzebne nie tylko w onCreate...
    private Button mPlayButton;
    private Button mPauseButton;

    private MediaPlayer mMediaPlayer;
    private int mCurrentTime = 0;
    private int mTotalTime = 0;
    // o ile przewijają przyciski przewijania (ms)
    private final static int DELTA_T = 10000;
```

# Multimedia – główna aktywność, obsługa paska przewijania

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    //ustawienie Layoutu...  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    //odczytanie referencji do etykiet...  
    mCurrentTimeTextView = findViewById(R.id.currentTimeTextView);  
    mTotalTimeTextView = findViewById(R.id.totalTimeTextView);  
  
    mSeekBar = findViewById(R.id.seekBar);  
    mSeekBar.setOnSeekBarChangeListener(  
        new SeekBar.OnSeekBarChangeListener() {  
            @Override  
            public void onStopTrackingTouch(SeekBar seekBar) {}  
  
            @Override  
            public void onStartTrackingTouch(SeekBar seekBar) {}  
        }  
    );  
}
```

# Multimedia – główna aktywność, obsługa paska przewijania i przycisków

```
// jeżeli użytkownik przesunie pasek przewijania
// musimy ustawić nowy czas na etykiecie i w
// otwarzaczu
@Override
public void onProgressChanged(SeekBar seekBar,
    int progress, boolean fromUser) {
    newTimeOnSeekBar(progress, fromUser);
})
// standardowe ustawienie obsługi przycisków odtwarzania,
// pauzy i przewijania...
mPlayButton = findViewById(R.id.playButton);
mPlayButton.setOnClickListener(v -> play());
mPauseButton = findViewById(R.id.pauseButton);
mPauseButton.setOnClickListener(v -> pause());

Button fastForwardButton = findViewById(R.id.fastForwardButton);
fastForwardButton.setOnClickListener(v -> fastForward());

Button rewindButton = findViewById(R.id.rewindButton);
rewindButton.setOnClickListener(v -> rewind());
```

# Multimedia – główna aktywność, przygotowanie odtwarzacza

```
// tworzenie odtwarzacza na postawie muzyki w zasobach (można też
// podać URI)
mMediaPlayer = MediaPlayer.create(this, R.raw.cool_tune);
// odtwarzacz nie jest gotowy do pracy natychmiast - trzeba
// poczekać
mMediaPlayer.setOnPreparedListener(mediaPlayer -> {
    // gdy jest gotowy możemy ustawić całkowity czas na etykiecie
    mTotalTime = mediaPlayer.getDuration();
    mTotalTimeTextView.setText(readableTime(mTotalTime));
    // i skonfigurować pasek
    mSeekBar.setClickable(true);
    mSeekBar.setMax(mTotalTime);
    mSeekBar.setProgress(mcurrentTime);
    // teraz można uruchomić odtwarzanie
    mPlayButton.setEnabled(true);
});
// co ma się stać gdy przestaną grać muzyka
mMediaPlayer.setOnCompletionListener(mediaPlayer -> {
    mCurrentTime = 0;
    timeUpdate(mcurrentTime);
    pause();
});
}
```

# Multimedia – główna aktywność, obsługa paska przewijania , Handler

```
// metoda pomocnicza ustawiająca nowy czas z paska w odtwarzaczu
// i na etykietce
private void newTimeOnSeekBar(int progress, boolean fromUser) {
    if (fromUser) {
        mMediaPlayer.seekTo(progress);
        mCurrentTime = progress;
        mCurrentTimeTextView.setText(readableTime(progress));
    }
    Log.d(TAG, "Progress on seek bar updated: " + progress);
}

// handler posłuży nam do aktualizowania paska postępu w miarę
// odtwarzania muzyki
private Handler handler = new Handler(Looper.getMainLooper());
```

# Multimedia – główna aktywność, obsługa przycisków, aktualizacja czasu

```
// metoda pomocnicza obsługująca przycisk włączający muzykę
private void play() {
    mMediaPlayer.start();
    mPauseButton.setEnabled(true);
    mPlayButton.setEnabled(false);
    // uruchomienie aktualizacji paska postępu
    // timeUpdate - obiekt klasy implementującej Runnable
    // pierwsza aktualizacja za sekundę
    handler.postDelayed(timeUpdate, 1000);
}

Runnable timeUpdate = new Runnable() {
    public void run() {
        // odczytujemy aktualną pozycję odtwarzacza
        int playerTime = mMediaPlayer.getCurrentPosition();
        Log.d(TAG, "Time update: " + playerTime);
        // i aktualizujemy pasek oraz etykietę
        timeUpdate(playerTime);
        // jeżeli dalej gra muzyka planujemy następną aktualizację paska
        if (mMediaPlayer.isPlaying())
            handler.postDelayed(timeUpdate, 1000);
    }
};
```

# Multimedia – główna aktywność, obsługa przycisków

```
//metoda pomocnicza wstrzymująca muzykę
private void pause() {
    mMediaPlayer.pause();
    mPauseButton.setEnabled(false);
    mPlayButton.setEnabled(true);
}

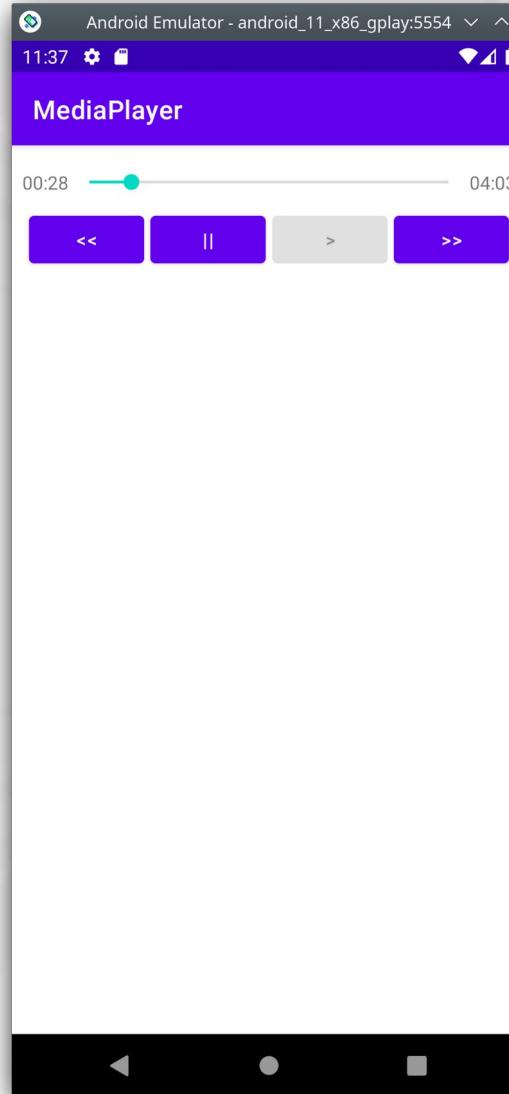
//metoda pomocnicza ustawiająca czas na etykiecie i pasku przewijania
private void timeUpdate(int time) {
    mCurrentTimeTextView.setText(readableTime(time));
    mSeekBar.setProgress(time);
}

//metoda pomocnicza obsługująca przycisk przewijania do przodu
private void fastForward() {
    //zabezpieczenie przed wyjściem poza muzyczkę
    if (mcurrentTime + DELTA_T > mTotalTime)
        mcurrentTime = mTotalTime;
    else
        mcurrentTime = mcurrentTime + DELTA_T;
    mMediaPlayer.seekTo(mcurrentTime);
    timeUpdate(mcurrentTime);
}
```

# Multimedia – główna aktywność, obsługa przycisków, formatowanie czasu

```
//metoda pomocnicza obsługująca przycisk przewijania do tyłu  
//analogiczna do przewijania do przodu...  
private void rewind() {  
    if (mCurrentTime - DELTA_T < 0)  
        mCurrentTime = 0;  
    else  
        mCurrentTime = mCurrentTime - DELTA_T;  
    mMediaPlayer.seekTo(mCurrentTime);  
    timeUpdate(mCurrentTime);  
}  
  
//metoda konwertująca czas w ms na zapis minuty:sekundy  
private String readableTime(int time) {  
    return String.format(  
        "%02d:%02d",  
        TimeUnit.MILLISECONDS.toMinutes(time),  
        TimeUnit.MILLISECONDS.toSeconds(time)  
        - TimeUnit.MINUTES.toSeconds(  
            TimeUnit.MILLISECONDS.toMinutes(time)));  
}  
}
```

# Multimedia





# Programowanie Aplikacji Mobilnych na Platformę Android

## Wykład 11 – Wstęp do grafiki na platformie Android.

Jakub Smołka



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



# Rysowanie w Androidzie

- Istnieją dwie podstawowe metody:
  - Metoda 1:
    - stworzenie własnego widoku np. dziedziczącego po klasie View
    - rysowanie grafiki/klatki animacji animacji na widoku umieszczonym w układzie elementów wewnętrz standardowej metody `onDraw()`
    - za uruchomienie rysowania grafiki odpowiedzialny jest normalny proces rysowania hierarchii elementów. Metody `invalidate()` (lub `postInvalidate()`) powiadamiają system, że widok należy narysować na nowo. System decyduje kiedy faktycznie narysować komponent
    - metoda zalecana do prostej, nie zmieniającej się zbyt często grafiki

# Rysowanie w Androidzie

- Metoda 2:
  - stworzenie widoku dziedziczącego po SurfaceView i implementacja SurfaceHolder.Callback
  - interfejs SurfaceHolder.Callback umożliwia reagowanie na utworzenie/zmiany/zniszczenie powierzchni do rysowania
  - rysowanie odbywa się po samodzielnym zablokowaniu kanwy/fragmentu kanwy w dowolnym momencie. Metoda rysująca np. onDraw() jest wywoływana samodzielnie przez twórcę aplikacji (nie przez system)
  - to podejście zalecane jest w wypadku, gdy grafika jest regularnie odświeżana (np. do animacji)

# Podstawowe klasy

- W systemie Android dostępnych jest wiele klas odpowiedzialnych za tworzenie grafiki
- Trzy podstawowe z nich to:
  - Klasa Bitmap – stanowiąca pojemnik na piksele
  - Klasa Canvas – zawiera metody rysujące prymitywy (podstawowe elementy obrazu: linie, prostokąty, koła itp. ale też bardziej skomplikowane np. ścieżki) na bitmapie
  - Klasa Paint – opisuje kolor (w tym przeźroczystość) i styl rysowanych elementów (prymitywów, tekstów, bitmap...)

# Klasa Canvas

- Klasa Canvas udostępnia interfejs do rysowania na obszarze, na którym powstaje obraz
- Obiekt klasy Canvas można uzyskać na kilka sposobów:
  - Jako parametr metody `onDraw()`. Odpowiedni obiekt tworzony jest automatycznie
  - W przypadku wykorzystania `SurfaceView` – za pomocą metody `lockCanvas()` obiektu klasy `SurfaceHolder`
  - Poprzez samodzielne utworzenie kanwy:

```
Bitmap b = Bitmap.createBitmap(100, 100,  
        Bitmap.Config.ARGB_8888);  
  
Canvas c = new Canvas(b);
```

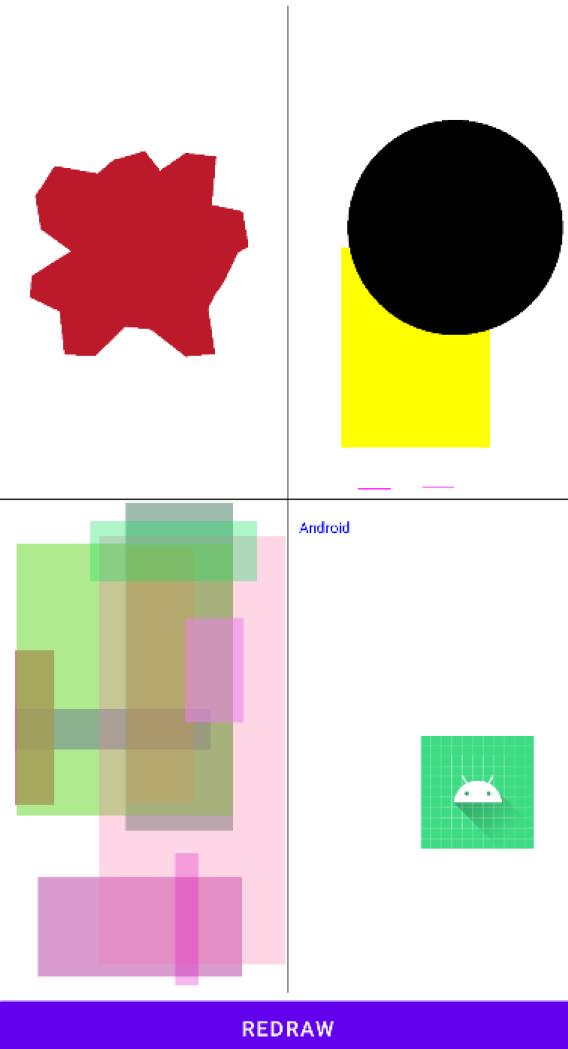
# Klasa Canvas – podstawowe metody

- `drawARGB(a,r,g,b)` / `drawColor(c)` – wypełnia całą kanwę określonym kolorem (osobne składowe lub stała zdefiniowana w klasie Color)
- `drawLine(x1,y1,x2,y2,paint)` – rysuje linię za pomocą określonej farby
- `drawRect(...,farba)` – Rysuje prostokąt za pomocą określonej farby (różne sposoby określania prostokąta: współrzędne, obiekt Rect)
- `drawPath(sciezka,farba)` – Rysuje ścieżkę (określoną przez obiekt Path) za pomocą określonej farby
- `drawCircle(x,y,r,farba)` – Rysuje koło o środku (x,y) i promieniu r
- `drawBitmap(bitmap,...,farba)` – Umieszcza bitmapę (lub jej fragment) w określonym miejscu (obszarze) kanwy (możliwe wycinanie/skalowanie)

# Klasa Paint – podstawowe metody

- `setARGB(a, r, g, b)/setColor(c)` – ustawia zadany kolor rysowania
- `setStyle(Paint.Style s)` – ustawia styl rysowania (FILL – tylko wypełnienie, STROKE – tylko kontur, FILL\_AND\_STROKE – kontur i wypełnienie)
- `setTextSize(float r)` – ustawia rozmiar tekstu (>0)
- `setAntiAlias(boolean aa)` – włącza / wyłącza (w zależności od parametru) anti-aliasing
- `setStrokeCap(Paint.Cap c)` – ustawia wygląd zakończenia linii (BUTT – równo z końcem ścieżki, ROUND – zaokrąglone, SQUARE – koniec ścieżki to środek kwadratu)
- `setStrokeWidth(float sz)` – ustawia szerokość konturu

# Własny komponent (metoda 1) – layout



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget
    .ConstraintLayout
    <!-- ... -->

    <!-- ważne jest view małą literą,
        jeżeli napisze się View używana jest
        klasa View a nie własny komponent
        podany w atrybucie class -->

<view
    android:id="@+id/paintView"
    class=
        "com.example.w11_custom_view_and.PaintView"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toTopOf=
        "@+id/redrawButton"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf=
        "parent"
    app:layout_constraintTop_toTopOf="parent">
</view>
</androidx.constraintlayout.widget
    .ConstraintLayout>
```

# Własny komponent (metoda 1) – główna aktywność

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        //standardowy początek...  
        Button redrawButton=findViewById(R.id.redrawButton);  
        redrawButton.setOnClickListener(v -> {  
            PaintView paintView=findViewById(R.id.paintView);  
            paintView.invalidate();  
        });  
    }  
}
```

# Własny komponent (metoda 1) – widok, konstruktory

```
public class PaintView extends View {  
  
    // konstruktor stosowany przy dodawaniu widoku w kodzie  
    public PaintView(Context context) {  
        super(context);  
        init();  
    }  
  
    // konstruktory wykorzystywane przy korzystaniu z XMLa  
    public PaintView(Context context, AttributeSet attrs) {  
        super(context, attrs);  
        init();  
    }  
  
    public PaintView(Context context, AttributeSet attrs,  
                    int defStyleAttr) {  
        super(context, attrs, defStyleAttr);  
        init();  
    }  
}
```

# Własny komponent (metoda 1) – widok, inicjalizacja

```
// obiekty których będziemy ciągle używać tworzymy raz (dla  
// poprawienia wydajności)  
private Paint mPaint;  
private Random mRandom;  
private Drawable mVectorForegroundShape;  
private Drawable mVectorBackgroundShape;  
  
private static final int SHAPE_SIZE = 200;  
  
private void init() {  
    mPaint = new Paint();  
    mRandom = new Random();  
    //standardowa wektorowa ikonka aplikacji (ma dwie części:  
    //tło i pierwszy plan)  
    mVectorForegroundShape = ContextCompat.getDrawable(  
        getContext(), R.drawable.ic_launcher_foreground);  
    mVectorBackgroundShape = ContextCompat.getDrawable(  
        getContext(), R.drawable.ic_launcher_background);  
}
```

# Własny komponent (metoda 1) – widok, rysowanie

```
// metoda wywoływana za każdym razem gdy trzeba narysować widok
@Override
protected void onDraw(Canvas canvas) {
    //zamalowanie na biało
    mPaint.setColor(Color.BLACK);
    // wyświetlenie linii dzielących komponent na 4 części
    int centerX = getWidth() / 2;
    int centerY = getHeight() / 2;
    canvas.drawColor(Color.WHITE);
    canvas.drawLine(centerX, 0, centerX, getHeight(), mPaint);
    canvas.drawLine(0, centerY, getWidth(), centerY, mPaint);
    // metody pomocnicze demonstrujące różne rzeczy
    pathDemo(canvas);
    shapesDemo(canvas);
    alphaDemo(canvas);
    textAndVectorShapeDemo(canvas);
}
```

# Własny komponent (metoda 1) – widok, ścieżki

```
// rysowanie niestandardowej ścieżki
private void pathDemo(Canvas canvas) {
    int centerX = getWidth() / 4;
    int centerY = getHeight() / 4;
    int angle = 0;
    // wybranie mniejszego promienia (szerokość i wysokość są być
    // różne - wybieramy krótszy promień)
    int radius = 2 * centerX / 3;
    if (radius > 2 * centerY / 3)
        radius = 2 * centerY / 3;
    // utworzenie ścieżki
    Path path = new Path();
    // ustawienie początku ścieżki
    path.moveTo(centerX + radius, centerY);
    // losowo zwiększamy kąt (od 10 do 19 stopni)
    angle += 10 + mRandom.nextInt(10);
    while (angle < 360) {
        // dodajemy kolejny punkt
        int tmpRadius = 2 * radius / 3
            + mRandom.nextInt(2 * radius / 3);
```

# Własny komponent (metoda 1) – widok, ścieżki

```
    float tmpX = (float) (centerX + tmpRadius
                           * Math.cos(Math.toRadians(angle)));
    float tmpY = (float) (centerY + tmpRadius
                           * Math.sin(Math.toRadians(angle)));
    path.lineTo(tmpX, tmpY);
    angle += 10 + mRandom.nextInt(10);
}
// aby ścieżka była zamknięta
path.close();
// 255 <=> nieprzeźroczysty, reszta losowe RGB
mPaint.setARGB(255, mRandom.nextInt(256), mRandom.nextInt(256),
                mRandom.nextInt(256));
canvas.drawPath(path, mPaint);
}

// tablica predefiniowanych kolorów
private int[] mColors = {Color.BLACK, Color.BLUE, Color.CYAN,
                        Color.GRAY, Color.GREEN, Color.MAGENTA, Color.MAGENTA,
                        Color.RED, Color.YELLOW};
```

# Własny komponent (metoda 1) – widok, kształty

```
private void shapesDemo(Canvas canvas) {  
    int minX = getWidth() / 2;  
    int minY = 0;  
    int width = getWidth() / 2;  
    int height = getHeight() / 2;  
  
    // rysowanie linii  
    mPaint.setColor(mColors[mRandom.nextInt(mColors.length)]);  
    int startX = minX + mRandom.nextInt(width);  
    int startY = minY + mRandom.nextInt(height);  
    int endX = minX + mRandom.nextInt(width);  
    int endY = minY + mRandom.nextInt(height);  
    canvas.drawLine(startX, startY, endX, endY, mPaint);  
  
    // rysowanie prostokąta  
    mPaint.setColor(mColors[mRandom.nextInt(mColors.length)]);  
    startX = minX + mRandom.nextInt(4 * width / 5);  
    startY = minY + mRandom.nextInt(4 * height / 5);  
    endX = startX + 10 + mRandom.nextInt(width - (startX - minX) - 10);  
    endY = startY + 10 + mRandom.nextInt(height - (startY - minY) - 10);
```

# Własny komponent (metoda 1) – widok, kształty, przeźroczystość

```
Rect rectangle = new Rect(startX, startY, endX, endY);
canvas.drawRect(rectangle, mPaint);

// rysowanie koła
mPaint.setColor(mColors[mRandom.nextInt(mColors.length)]);
int maxRadius = width / 2;
if (maxRadius > height / 2)
    maxRadius = height / 2;
int radius = mRandom.nextInt(maxRadius);
startX = minX + radius + mRandom.nextInt(width - 2 * radius);
startY = minY + radius + mRandom.nextInt(height - 2 * radius);
canvas.drawCircle(startX, startY, radius, mPaint);
}

// demo przeźroczystości
private void alphaDemo(Canvas canvas) {
    int minX = 0;
    int minY = getHeight() / 2;
    int width = getWidth() / 2;
    int height = getHeight() / 2;
```

# Własny komponent (metoda 1) – widok, przeźroczystość, tekst, zasoby wektorowe

```
for (int i = 0; i < 10; ++i) {
    int startX = minX + mRandom.nextInt(4 * width / 5);
    int startY = minY + mRandom.nextInt(4 * height / 5);
    int endX = startX + 10
        + mRandom.nextInt(width - (startX - minX) - 10);
    int endY = startY + 10
        + mRandom.nextInt(height - (startY - minY) - 10);
    Rect rectangle = new Rect(startX, startY, endX, endY);
    mPaint.setARGB(mRandom.nextInt(256), mRandom.nextInt(256),
                   mRandom.nextInt(256), mRandom.nextInt(256));
    canvas.drawRect(rectangle, mPaint);
}
}

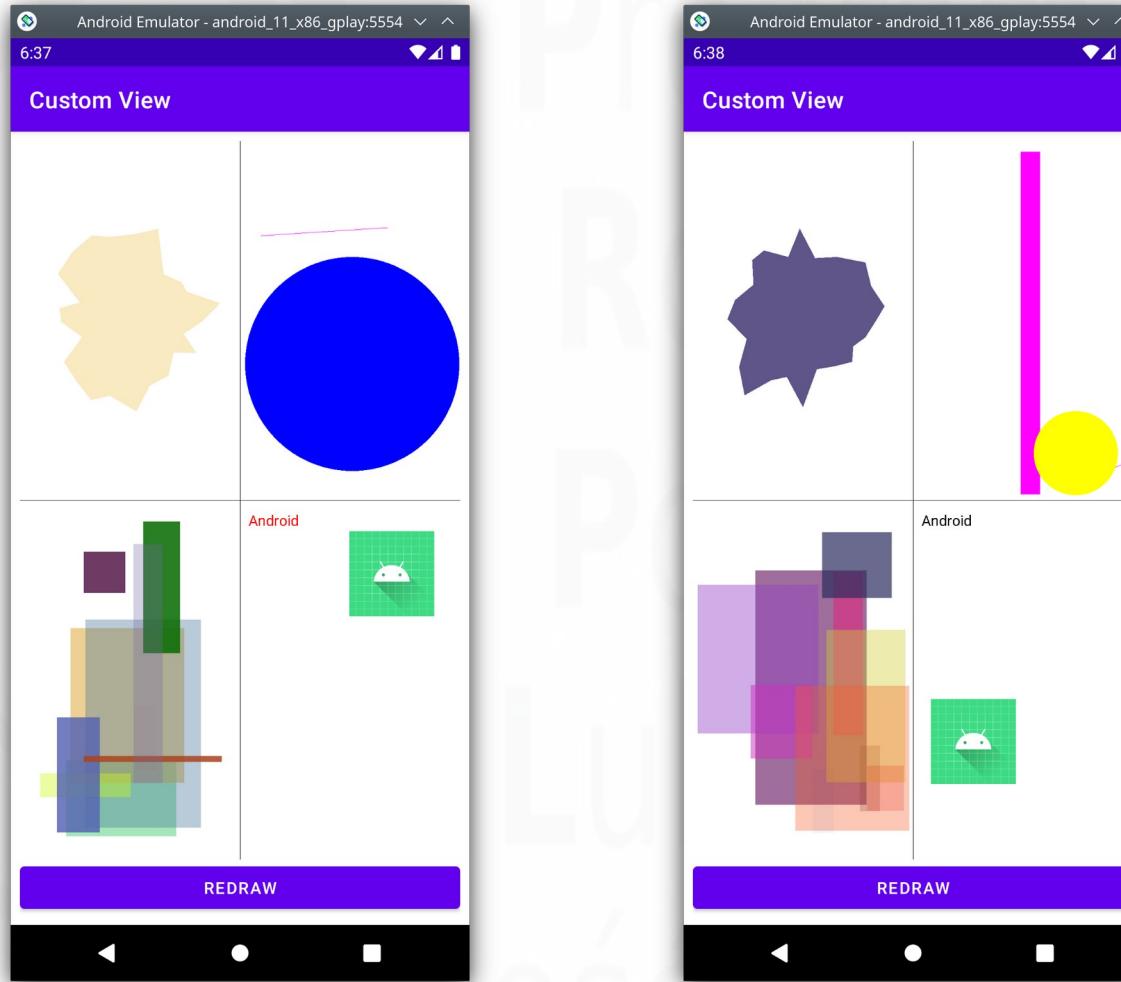
// demo wyświetlania tekstu i rysowania bitmap
private void textAndVectorShapeDemo(Canvas canvas) {
    int minX = getWidth() / 2;
    int minY = getHeight() / 2;
    int width = getWidth() / 2;
    int height = getHeight() / 2;
```

# Własny komponent (metoda 1) – widok, tekst, zasoby wektorowe

```
mPaint.setColor(Color.BLACK); // żeby wyłączyć przeźroczystość
// rysujemy ikonkę programu w losowych współrzędnych
int iconX = minX + mRandom.nextInt(width - SHAPE_SIZE);
int iconY = minY + mRandom.nextInt(height - SHAPE_SIZE);
mVectorBackgroundShape.setBounds(iconX, iconY,
    iconX + SHAPE_SIZE, iconY + SHAPE_SIZE);
mVectorForegroundShape.setBounds(iconX, iconY,
    iconX + SHAPE_SIZE, iconY + SHAPE_SIZE);
mVectorBackgroundShape.draw(canvas);
mVectorForegroundShape.draw(canvas);

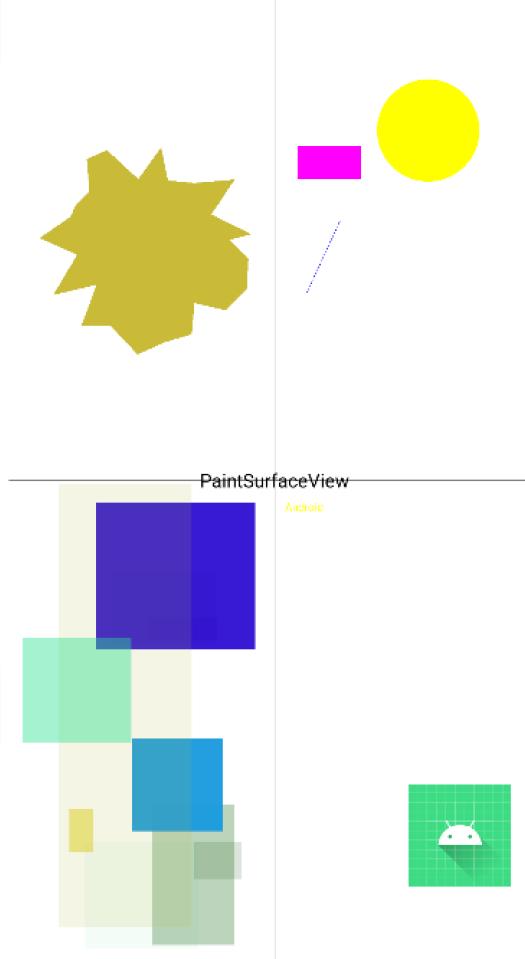
// wyświetlenie napisu o losowej wielkości i kolorze
mPaint.setColor(mColors[mRandom.nextInt(mColors.length)]);
mPaint.setTextSize(20+mRandom.nextInt(40));
canvas.drawText("Android", minX + 20, minY + 60, mPaint);
}
```

# Własny komponent (metoda 1)



# Komponent SurfaceView (metoda 1+2)

## – layout



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget
    .ConstraintLayout
    <!-- inny sposób (zamiast używania
        znacznika <view> -->

    <com.example.w11_custom_drawing_and
        .PaintSurfaceView
        android:id="@+id/paintSurfaceView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf=
            "parent"
        app:layout_constraintEnd_toEndOf=
            "parent"
        app:layout_constraintStart_toStartOf=
            "parent"
        app:layout_constraintTop_toTopOf=
            "parent" />

</androidx.constraintlayout.widget
    .ConstraintLayout>
```

# Komponent SurfaceView (metoda 1+2)

- własny komponent , konstruktory
- Główna aktywność tylko ustawia layout. Cała logika zawarta jest w widoku dziedziczącym po SurfaceView

```
public class PaintSurfaceView extends SurfaceView implements  
    SurfaceHolder.Callback {  
  
    public PaintSurfaceView(Context context) {  
        super(context);  
        init();  
    }  
    public PaintSurfaceView(Context context, AttributeSet attrs) {  
        super(context, attrs);  
        init();  
    }  
    public PaintSurfaceView(Context context, AttributeSet attrs,  
        int defStyle) {  
        super(context, attrs, defStyle);  
        init();  
    }  
}
```

# Komponent SurfaceView (metoda 1+2)

## – własny komponent, inicjalizacja

```
//obiekty których będziemy ciągle używać tworzymy raz (dla  
//poprawienia wydajności)  
private Paint mPaint;  
private Random mRandom;  
private Drawable mVectorForegroundShape;  
private Drawable mVectorBackgroundShape;  
  
private static final int SHAPE_SIZE = 200;  
  
private void init() {  
    //tworzenie farby, generatora i kształtów...  
    mPaint = new Paint();  
    mRandom = new Random();  
    //standardowa wektorowa ikonka aplikacji (ma dwie części: tło i  
    //pierwszy plan)  
    mVectorForegroundShape = ContextCompat.getDrawable(getContext(),  
        R.drawable.ic_launcher_foreground);  
    mVectorBackgroundShape = ContextCompat.getDrawable(  
        getContext(),  
        R.drawable.ic_launcher_background);
```

# Komponent SurfaceView (metoda 1+2)

- własny komponent, wątek rysujący

```
//rejestracja listenera
getHolder().addCallback(this);
//WAŻNE - bez tego się nie rysuje
//ustawiamy na false ponieważ definiujemy onDraw()
//tzn. informujemy, że widok sam też coś rysuje
setWillNotDraw(false);
}

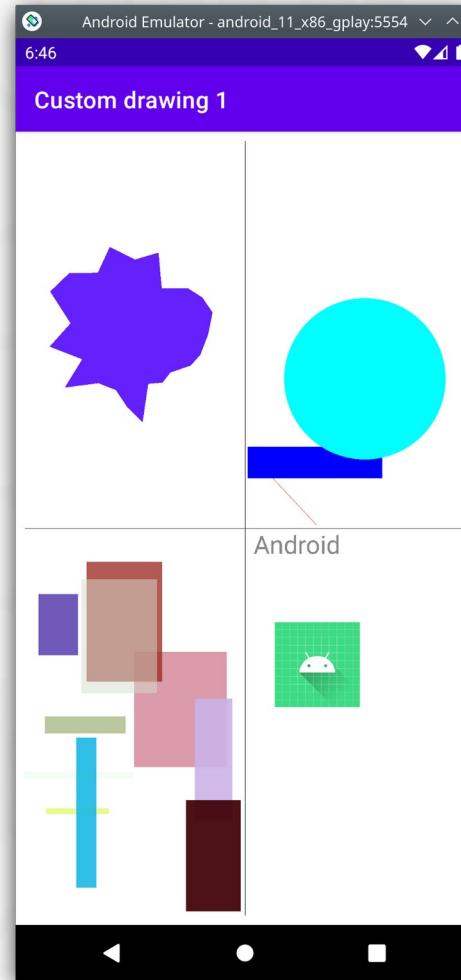
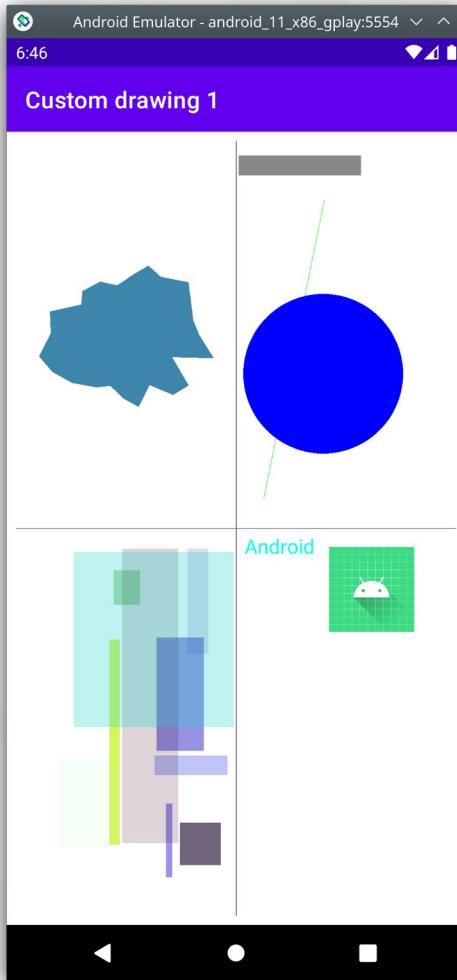
//wątek aktualizujący powierzchnię
private class DrawingThread extends Thread {
    public boolean mThreadRunning = true;

    public void run() {
        while (mThreadRunning) {
            //postInvalidate() zamiast invalidate()
            //bo z innego wątku
            postInvalidate();
            try {
                Thread.sleep(100); //10 klatek na sekundę
            } catch (InterruptedException e) {
        }}}}
```

# Komponent SurfaceView (metoda 1+2) – własny komponent, SurfaceHolder.Callback

```
private DrawingThread mDrawingThread;
//implementacja interfejsu SurfaceHolder.Callback
@Override
public void surfaceCreated(SurfaceHolder holder) {
    mDrawingThread = new DrawingThread();
    mDrawingThread.start();
}
@Override
public void surfaceChanged(SurfaceHolder holder, int format,
    int width, int height) { }
@Override
public void surfaceDestroyed(SurfaceHolder holder) {
    //ładnie kończymy wątek
    mDrawingThread.mThreadRunning = false;
    try {
        mDrawingThread.join();
    } catch (InterruptedException e) { }
}
//metoda onDraw() i metody ???Demo() takie same jak w poprzednim
//przykładzie ...
}
```

# Komponent SurfaceView (metoda 1+2) – własny komponent



# Komponent SurfaceView (metoda 2) – własny komponent,inicjalizacja

- Główna aktywność tylko ustawia layout. Layout aktywności jest identyczny jak w poprzednim przykładzie. Cała logika zawarta jest w widoku dziedziczącym po SurfaceView

```
public class PaintSurfaceView extends SurfaceView implements  
SurfaceHolder.Callback {  
    //Pola i konstruktory takie same jak poprzednio...  
    private SurfaceHolder mSurfaceHolder;  
    private void init() {  
        //początek taki sam jak w poprzednim przykładzie...  
        //rejestracja listenera  
        mSurfaceHolder = getHolder();  
        mSurfaceHolder.addCallback(this);  
        // Teraz nie musimy wywoływać setWillNotDraw()  
        // Aby otrzymywać informacje o naciśnięciu klawiszy  
        // włączamy „fokusowalność” widoku  
        setFocusable(true);  
    }  
}
```

# Komponent SurfaceView (metoda 2) – własny komponent, wątek rysujący

```
public Boolean mThreadRunning = true;
// watek aktualizujący powierzchnię
private class DrawingThread extends Thread {
    // rysowanie jak w oficjalnych przykładach Google
    public void run() {
        while (mThreadRunning) {
            Canvas canvas = null;
            try {
                // zwraca kanwę, na której można rysować, każdy
                // piksel kanwy w prostokącie przekazanym jako
                // parametr musi być narysowany od nowa
                // inaczej: rozpoczęcie edycji zawartości kanwy
                canvas = mSurfaceHolder.lockCanvas(null);
                // sekcja krytyczna - żaden inny wątek nie może
                // używać
                // pojemnika (<=>rysować)
                synchronized (mSurfaceHolder) {
                    // sekcja krytyczna - żaden inny wątek nie
                    // może zmienić flagi mThreadRunning
                    synchronized (mThreadRunning) {
                        if (mThreadRunning)
                            drawFrame(canvas);
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                if (canvas != null)
                    mSurfaceHolder.unlockCanvasAndPost(canvas);
            }
        }
    }
}
```

# Komponent SurfaceView (metoda 2) – własny komponent, wątek rysujący

```
        } finally {
            // w bloku finally - gdyby wystąpił wyjątek w
            // powyższym powierzchnia zostanie zostawiona w
            // spójnym stanie
            if (canvas != null) {
                // koniec edycji kanwy i wyświetlenie
                // rysunku na ekranie
                mSurfaceHolder.unlockCanvasAndPost(canvas);
            }
        }
        try {
            Thread.sleep(100); // 10 klate na sekundę
        } catch (InterruptedException e) {
    }
}
private DrawingThread mDrawingThread;
```

# Komponent SurfaceView (metoda 2) – własny komponent, sterowanie animacją

```
private void startAnimation() {  
    // uruchamiamy animację w momencie utworzenia powierzchni  
    synchronized (mThreadRunning) {  
        mThreadRunning = true;  
    }  
    mDrawingThread = new DrawingThread();  
    mDrawingThread.start();  
}  
  
private void stopAnimation() {  
    // ładnie kończymy wątek  
    synchronized (mThreadRunning) {  
        mThreadRunning = false;  
    }  
    try {  
        mDrawingThread.join();  
    } catch (InterruptedException ignored) {  
    }  
}
```

# Komponent SurfaceView (metoda 2) – własny komponent, sterowanie animacją

```
private boolean mStopThread = true;
//metoda pomocnicza wstrzymująca lub uruchamiającą animację
private void pauseResume() {
    if (mStopThread) {
        stopAnimation();
        mStopThread = false;
    } else {
        startAnimation();
        mStopThread = true;
    }
}
//interfejs SurfaceHolder.Callback
@Override
public void surfaceCreated(SurfaceHolder holder) {
    startAnimation();
}
@Override
public void surfaceChanged(SurfaceHolder holder, int format,
    int width, int height) { }
@Override
public void surfaceDestroyed(SurfaceHolder holder) {
    stopAnimation();
}
```

# Komponent SurfaceView (metoda 2) – własny komponent, obsługa klawiatury

```
//obsługa klawiatury
@Override
public boolean onKeyUp(int keyCode, KeyEvent event) {
    // spacja zatrzymuje animację
    if (keyCode == KeyEvent.KEYCODE_SPACE)
        pauseResume();
    return super.onKeyUp(keyCode, event);
}
//onTouchEvent i performClick należy implementować razem
@Override
public boolean onTouchEvent(MotionEvent event) {
    performClick();
    return super.onTouchEvent(event);
}
@Override
public boolean performClick() {
    pauseResume();
    return super.performClick();
}
protected void drawFrame(Canvas canvas) {
    //treść metody taka sama jak onDraw() w poprzednich przykładach...
}
//metody ???Demo() takie same jak w poprzednich przykładach...
}
```



# Programowanie Aplikacji Mobilnych na Platformę Android

## Wykład 12 – Różnice pomiędzy aplikacjami dla różnych typów urządzeń.

Jakub Smołka



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



# Podstawowe różnice pomiędzy urządzeniami

- Urządzenia z systemem Android są bardzo różnorodne
- Głównymi cechami, którymi różnią się urządzenia są
  - *rozmiar ekranu*
  - *proporcje ekranu*
- Dodatkowo (w wielu wypadkach) może zmieniać się orientacja urządzenia
- Aplikacja może być również uruchamiana w trybie *multi-window*
- Dlatego layouty aplikacji powinny być *responsywne* i *adaptacyjne*

# Klasy rozmiarów okna

- Rozmiar okna został podzielony na 3 kategorie
- Istnieją 3 kategorie dla szerokości i 3 kategorie dla wysokości
- Szerokość: *kompaktowa/compact* (<600dp), *średnia/medium* ( $\geq 600dp$  i  $< 840dp$ ), *rozszerzona/expanded* ( $\geq 840dp$ )
- Wysokość: *kompaktowa/compact* (<480dp), *średnia/medium* ( $\geq 480dp$  i  $< 900dp$ ), *rozszerzona/expanded* ( $\geq 900dp$ )
- Dostępna szerokość jest często ważniejsza niż dostępna wysokość (przewijanie w pionie jest powszechnie stosowane)
- Wymagania dla kolejnych kategorii szerokości spełnia: prawie 100% telefonów w pionie, prawie 94% tabletów w pionie, 97% tabletów w poziomie

# Klasy rozmiarów okna

- Uwzględnienie szerokości ekranów jest wystarczające w większości aplikacji
- Wymagania dla kolejnych kategorii wysokości spełnia: prawie 100% telefonów, prawie 97% tabletów w poziomie i prawie 98% telefonów w pionie, 94% tabletów w pionie
- Dlaczego uwzględnia się rozmiar okna (nie ekranu)?
  - na urządzeniach mobilnych można podzielić ekran
  - na urządzeniach z *ChromeOS* aplikacje mogą działać w oknie a okno aplikacji może się zmieniać
  - w urządzeniach „składanych” rozmiar ekranu może się zmieniać

# Wspieranie różnych urządzeń

- Tworzenie elastycznych layoutów
  - ConstraintLayout pozwala na tworzenie responsywnych układów elementów
  - Preferowanie wartości wrap\_content albo match\_parent w przypadku wysokości/szerokości zamiast konkretnych wartości
- Tworzenie alternatywnych layoutów dla różnych urządzeń – pozwala stworzyć układ elementów optymalny dla różnych typów urządzeń
  - Alternatywne layouty posiadają kwalifikatory

# Wspieranie różnych urządzeń

- Typowe kwalifikatory
  - sw???dp np. sw600dp (*sw* = smallest width) – odnosi się do krótszego boku ekranu (niezależnie od ustawienia urządzenia)
  - w???dp np. w600dp (*w* = width) – odnosi się do dostępnej szerokości urządzenia (zależy od ustawienia urządzenia)
  - port – ustawienie pionowe
  - land – ustawienie poziome
  - Kwalifikatory można łączyć np. sw600dp-land
- Aplikację należy testować na ekranach o różnych proporcjach np. 21:9, 16:9, 5:3... (można określić *min* i *maxAspectRatio* aktywności w manifeście aplikacji)

# Android TV

- Aplikacje dla Android TV mają tę samą strukturę, co aplikacje na telefony i tablety – używa się tych samych technik tworzenia aplikacji
- Model interakcji z użytkownikiem w przypadku telewizora znacznie różni się od interakcji z telefonem czy tabletem
- Aplikacje dla telewizorów wymagają
  - zaprojektowania układów elementów, które będą łatwe do zrozumienia z odległości ok. 3 metrów
  - zapewnienia nawigacji, której można używać tylko za pomocą klawiatury kierunkowej i przycisku wyboru

# Android TV – manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.w12_tv_sample">
    <!-- ... -->
    <uses-feature
        android:name="android.hardware.touchscreen"
        android:required="false" />
    <uses-feature
        android:name="android.software.leanback"
        android:required="true" />
    <!-- ... -->
    <application
        <!-- ... -->
        <activity
            <!-- ... -->
            android:banner="@drawable/app_icon_your_company"
            android:logo="@drawable/app_icon_your_company"
            android:screenOrientation="Landscape">
                <intent-filter>
                    <action android:name="android.intent.action.MAIN" />
                    <category android:name=
                        "android.intent.category.LEANBACK_LAUNCHER" />
                </intent-filter>
            </activity>
            <!-- ... -->
        </application>
    </manifest>
```

# Android TV – manifest

- Poniższy wpis oznacza, że aplikacja wymaga z interfejsu *Leanback* stosowanego w Android TV. Atrybutu required na true oznacza, że aplikacja nie zadziała na telefonach/tabletach

```
<!-- ... -->
<uses-feature
    android:name="android.software.leanback"
    android:required="true" />
<!-- ... -->
```

- Poniższy wpis oznacza, że aplikacja nie wymaga ekranu dotykowego

```
<!-- ... -->
<uses-feature
    android:name="android.hardware.touchscreen"
    android:required="false" />
<!-- ... -->
```

# Android TV – manifest

- Aplikacja dla Android TV wymaga zadeklarowania aktywności telewizyjnej (kategoria LEANBACK\_LAUNCHER)
- W przypadku aktywności telewizyjnej wymagany jest banner (zasób 320x180px z kwalifikatorem xhdpi)
- W aplikacjach uniwersalnych (dla TV i telefonów/tabletów) nie należy używać tej samej aktywności

```
<activity
    <!-- ... -->
    android:banner="@drawable/app_icon_your_company"
    android:logo="@drawable/app_icon_your_company"
    android:screenOrientation="landscape">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name=
                "android.intent.category.LEANBACK_LAUNCHER" />
        </intent-filter>
    </activity>
```

# Android TV – sprzęt

- Android TV nie obsługuje następujących elementów określanych za pomocą znacznika `uses-feature`
  - `android.hardware.touchscreen` – ekran dotykowy
  - `android.hardware.faketouch` – emulator ekranu dotykowego
  - `android.hardware.telephony` – telefonia
  - `android.hardware.camera` – aparat
  - `android.hardware.nfc`
  - `android.hardware.location.gps`
  - `android.hardware.microphone`
  - `android.hardware.sensor` – sensory
  - `android.hardware.screen.portrait` – ekran w ustawieniu pionowym
- Oznaczenie powyższych cech jako wymagane w manifeście aplikacji spowoduje brak możliwości zainstalowania aplikacji na Android TV

# Android TV – sprzęt

- Użycie niektórych uprawnień implikuje wymaganie określonych cech sprzętowych np.
  - RECORD\_AUDIO (→ mikrofon), CAMERA (→ aparat), ACCESS\_FINE\_LOCATION (→ GPS), ACCESS\_WIFI\_STATE (→ wifi)
- Możliwe jest stworzenie aplikacji uniwersalnej, która w trakcie wykonania sprawdza na jakim urządzeniu działa i włącza / wyłącza odpowiednie funkcje

```
UiModeManager uiModeManager = (UiModeManager) getSystemService(UI_MODE_SERVICE);
if (uiModeManager.getCurrentModeType() == Configuration.UI_MODE_TYPE_TELEVISION) {
    Log.d(TAG, "Running on a TV");
} else {
    Log.d(TAG, "Running on a non-TV");
}
if (getPackageManager().hasSystemFeature(PackageManager.FEATURE_TELEPHONY)) {
    Log.d(TAG, "Can make phone calls");
}
```

# Android TV – sterowanie

- Urządzenia Android TV sterowane są za pomocą kontrolerów.
- Obsługa kontrolerów jest identyczna z obsługą klawiatury – generowane są zdarzenia opisane za pomocą KeyEvent
- Aplikacja powinna obsługiwać następujące zdarzenia w określony sposób:
  - BUTTON\_B, BACK – wstecz
  - BUTTON\_SELECT, BUTTON\_A, ENTER, DPAD\_CENTER, KEYCODE\_NUMPAD\_ENTER – wybór elementu
  - DPAD\_UP, DPAD\_DOWN, DPAD\_LEFT, DPAD\_RIGHT – nawigacja

# Android TV – Biblioteki AndroidX

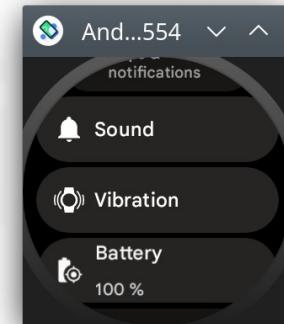
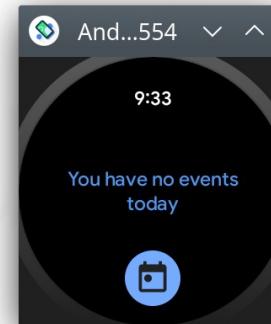
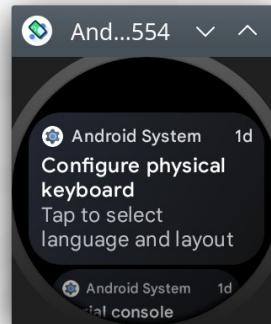
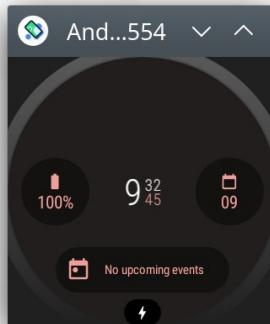
- AndroidX zawiera szereg bibliotek stworzonych z myślą o aplikacjach dla telewizorów
  - androidx.leanback.app – fragmenty implementujące elementy interfejsu aplikacji
  - androidx.leanback.database – przetwarzanie kursorów
  - androidx.leanback.graphics – modyfikacja granic (bounds) / kolorów elementów graficznych
  - androidx.leanback.media – klasy do odtwarzania mediów
  - androidx.leanback.preference – tworzenie ustawień aplikacji
  - androidx.leanback.system – udostępnia ustawienia Leanback
  - androidx.leanback.widget – komponenty interfejsu
  - androidx.leanback.widget.picker – komponenty wyboru dat, kolorów, czasu itp..

# Wear OS – najważniejsze koncepcje

- Aplikacje dla Wear OS powinny umożliwiać wykonywanie podstawowych zadań (zamiast zapewniać funkcjonalność odpowiednika na telefon)
- Dostępne funkcje powinny być łatwo dostępne aby można je było łatwo wykonać (nie powodować dyskomfortu u użytkownika - „optymalizacja dla nadgarstka”)
- Należy używać właściwych sposobów prezentacji danych (następny slajd)
- Należy umożliwić możliwość pracy offline (połączenie z internetem może nie być dostępne cały czas)
- Należy prezentować tylko istotne dla użytkownika informacje

# Wear OS – interfejs użytkownika

- W systemie Wear OS interfejs użytkownika jest projektowany wokół zadań wykonywanych na prezentowanych informacjach
  - *komplikacja (ang. complication)* – element prezentujący informację, którą użytkownik będzie sprawdzał wiele razy dziennie
  - *powiadomienie* – ważne informacje, zapewnia akcje kontekstowe
  - *kafelek (ang. tile)* – zapewniają szybki dostęp do ważnych informacji / akcji
  - *nakładki (ang. overlay)* – interaktywne przewijane listy



# Wear OS – manifest

- atrybut `standalone` – określa czy aplikacja może działać samodzielnie na zegarku czy wymaga aplikacji towarzyszącej na telefonie

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.w12_wear_sample_and">
    <!-- ... -->
    <uses-feature android:name="android.hardware.type.watch" />
    <application
        <!-- ... -->
        <uses-library
            android:name="com.google.android.wearable"
            android:required="true" />
        <meta-data
            android:name="com.google.android.wearable.standalone"
            android:value="true" />
        <activity
            <!-- ... -->
            </activity>
    </application>
</manifest>
```

# Wear OS – nawigacja

- W Wear OS standardowym gestem jest *przesuń, aby odrzucić*
  - gest polega na przesunięciu palcem od lewej do prawej strony
- Zalecane jest
  - stosowanie pionowych układów elementów (layoutów)
  - budowanie aplikacji tylko z aktywnościami, które obsługują gest automatycznie
  - nieimplementowanie w aplikacjach poziomych gestów
  - niestosowanie fragmentów (w których trzeba obsługę gestu odrzucania zaimplementować samodzielnie)

# Wear OS – biblioteki

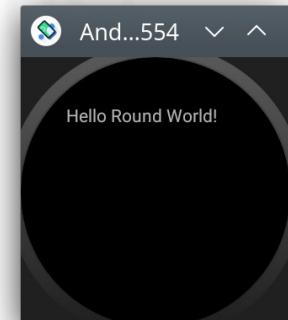
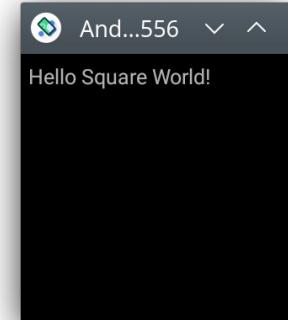
- Dostępna jest biblioteka *AndroidX* (w ramach *Android Jetpack*) – *Wear OS UI Library*

```
dependencies {  
    //...  
    //standardowe biblioteki WearOS  
    implementation 'androidx.wear:wear:1.2.0'  
    //dostęp do informacji o elementach sterujących np. przyciskach  
    implementation "androidx.wear:wear-input:1.1.0"  
}
```

- Biblioteka zawiera m.in. następujące elementy:
  - **CurvedTextView** – zakrzywiony tekst
  - **DismissibleFrameLayout** – układ, który umożliwia użytkownikowi odrzucenie dowolnego widoku poprzez naciśnięcie przycisku wstecz lub przesunięcie palcem
  - **WearableRecyclerView** – lista dostosowana do zegarków

# Wear OS – kształt ekranu

- Dostępne są layouty pozwalające na dostosowanie do kształtu ekranu:
  - BoxInsetLayout – stosuje wcięcia aby zawartość nie była przycięta na okrągłym ekranie. Za pomocą atrybutu layout\_boxedEdges można określić do, których krawędzi będą zastosowane wcięcia (np. left | top)
  - CurvedLayout – układ do wyświetlania pionowej listy elementów zoptymalizowaną pod kątem okrągłych ekranów
  - Układom dostosowanym do określonego typu ekranu nadaje się kwalifikatory round i notround



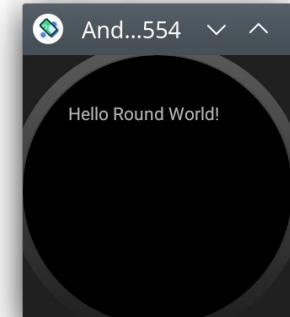
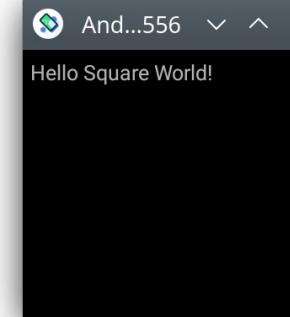
# Wear OS – kształt ekranu

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.wear.widget.BoxInsetLayout
    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"
    android:padding="@dimen/box_inset_layout_padding"
    tools:context=".MainActivity"
    tools:deviceIds="wear">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="@dimen/inner_frame_layout_padding"
        app:layout_boxedEdges="all">

        <TextView
            android:id="@+id/text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/hello_world" />

    </FrameLayout>
</androidx.wear.widget.BoxInsetLayout>
```



# Wear OS – dostęp do sieci

- Aplikacje Wear OS mogą korzystać z połączenia z Internetem
- Gdy zegarek ma połączenie Bluetooth z telefonem, ruch sieciowy zegarka jest zazwyczaj przesyłany przez telefon
- Gdy telefon jest niedostępny, używane są sieci Wi-Fi i komórkowe
- Wear OS automatycznie obsługuje przełączanie między sieciami
- Można używać protokołów, takich jak HTTP, TCP i UDP ale niektóre API nie są dostępne np. `android.webkit`
- Do wykonywania operacji asynchronicznych lub do okresowego odpytywania usług sieciowych zalecane jest korzystanie z `WorkManager API`
- Klasa `ConnectivityManager` umożliwia wykrywanie dostępu do sieci i żądanie połączenia z siecią o określonej przepustowości.

**POLITECHNIKA LUBELSKA**  
**WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI**  
**INFORMATYKA**



Materiały zostały opracowane w ramach projektu  
„Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga”,  
umowa nr **POWR.03.05.00-00-Z060/18-00**  
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020  
współfinansowanego ze środków Europejskiego Funduszu Społecznego



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny

