

ĆWICZENIE 1

Podstawy tworzenia aplikacji dla systemu Android

Mariusz Fraś

Cele ćwiczenia

1. Poznanie ogólnej, podstawowej architektury aplikacji Android.
 2. Poznanie podstawowych technik tworzenia aktywności i interfejsu graficznego aplikacji.
 3. Opanowanie pierwszej z podstawowych techniki nawigacji pomiędzy aktywnościami.
Zapoznanie się z mechanizmem intencji.
 4. Poznanie podstaw obsługi zdarzeń.
- **Pierwsza część ćwiczenia (Zadanie – część I)** jest instrukcją sposobu realizacji, implementacji funkcjonalności, do wykonania według podanych informacji (oraz ewentualnych poleceń prowadzącego zajęcia laboratoryjne).
 - **Druga część ćwiczenia (Zadanie – część II)** wymaga przygotowania i samodzielnej realizacji. Jest do prezentacji lub do wykonania na kolejnych zajęciach (wg polecenia prowadzącego).

1. Architektura aplikacji – aktywności, interfejs użytkownika, intencje

- *Materiał wymagany do laboratorium jest omawiany na wykładzie.*
- *Poleca się także m.in. witrynę <http://developer.android.com/>*
- *Jest także wiele innych witryn przedstawiających techniki programowania aplikacji Android (niektóre z nich będą podane przez prowadzącego zajęcia).*

Niektóre z nich:

- <https://codelabs.developers.google.com/?cat=android>
- <https://www.androidhive.info/>
- <https://www.vogella.com/tutorials/android.html>
- <https://guides.codepath.com/android/>
- <https://www.journaldev.com/android/>
- <https://stackoverflow.com/>

UWAGI OGÓLNE:

- We wszystkich ćwiczeniach podawane są podstawowe instrukcje dla tworzenia kodu. Kompilator może sygnalizować czasem błędy ze względu na brak jakiejś czynności – np. nie zaimportowanie potrzebnej biblioteki). We wszystkich przypadkach należy skorygować sygnalizowane błędy przez kompilator (np. importując odpowiednie klasy itp.) wykorzystując podpowiedzi kompilatora..
- W całej instrukcji wykrępowanie - ... - z reguły oznacza pominiętą w opisie część kodu.

2. Zadanie – część I.

Konstrukcja podstawowej aplikacji Android.

W zadanie polega na zrealizowaniu prostej aplikacji składającej się z okien z prostymi kontrolkami oraz zaimplementowanej prostej nawigacji między nimi.

2.1. Przygotowanie środowiska Android Studio

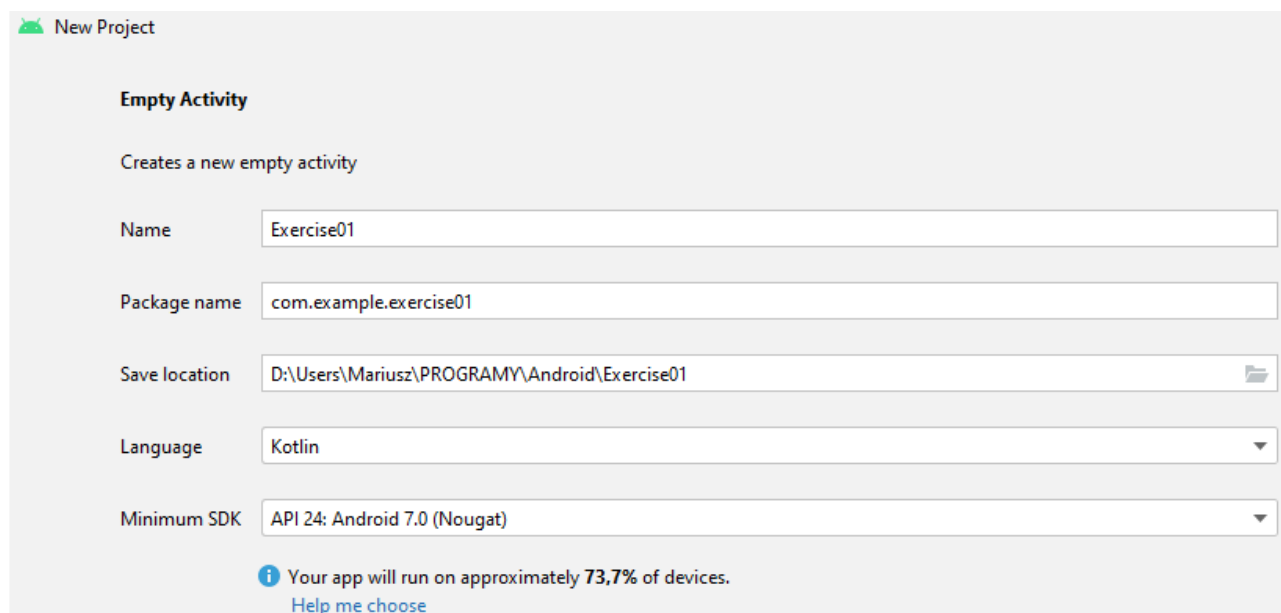
- Jeżeli nie zrobiono tego wcześniej, należy skonfigurować Android Studio wg instrukcji z **ćwiczenia wstępnego** (bez tworzenia aplikacji w tym ćwiczeniu).

2.2. Utworzenie projektu nowej aplikacji

- Opcja *New Project* pozwala wybrać projekty dla różnych środowisk (urządzeń) docelowych – m.in.: **Phone and Tablet**, **Wear OS**, **Television**, **Automotive** (Android Auto).
- Projekty dla Smartfonów:
 - wybieramy **Phone and Tablet**.,
 - Dostępne są predefiniowane projekty o różnym stopniu złożoności aplikacji – np.
 - **Empty Views Activity** – najmniej rozbudowany projekt (uwaga: NIE Empty Activity)
 - **Basic Views Activity** – projekt z kilkoma podstawowymi elementami działań w aplikacji (puste menu, tzw. FAB, itp.)
 - itd.

W ćwiczeniu wybieramy **Empty Views Activity**

- podajemy nazwę aplikacji,
- ewentualnie korygujemy nazwę pakietu,
- ustawiamy **Minimum API Level** (np. na **API 24** Android 7.0 albo nawet mniejsze),



New Project

Empty Activity

Creates a new empty activity

Name: Exercise01

Package name: com.example.exercise01

Save location: D:\Users\Mariusz\PROGRAMY\Android\Exercise01

Language: Kotlin

Minimum SDK: API 24: Android 7.0 (Nougat)

i Your app will run on approximately 73,7% of devices.
[Help me choose](#)

- Po zatwierdzeniu należy trochę poczekać aż środowisko zbuduje projekt.

- Domyślny projekt budowany jest w oparciu o biblioteki wsparcia dla niższych wersji Androida.

2.2.1. Przegląd projektu

- Przejrzyj zawartość projektu – pliki i ich zawartość w oknie projektu (po lewej stronie).
- Zwróć uwagę na zakładki pojawiające się u góry i u dołu okna zasobów xml. Przejrzyj je.
- Zwróć uwagę na **Gradle Scripts** – skrypty używane do konfigurowania i kompilacji projektu, w tym *build.gradle (Module app)*.
- Zwróć uwagę na zawartość plików **xml** i odwołania do zasobów definiowanych w innych plikach xml. W tym plik manifestu **AndroidManifest.xml**, plik układu graficznego aktywności (w węźle *app\res\layout*) **activity_main.xml** i **strings.xml** z tekstami do wyświetlania.

2.2.2. Pliki skryptów Gradle i AndroidManifest.xml (ustawienia elementów budowy aplikacji)

- W okienku *Project* (po lewej stronie) węzeł *Gradle Scripts* → *build.gradle (Module app)* (otwierany przez podwójne kliknięcie), podsekcja *defaultConfig* (w sekcji *android*) zawiera m.in. wersję minimalną i docelową SDK.

Może być ona ustawiona następująco:

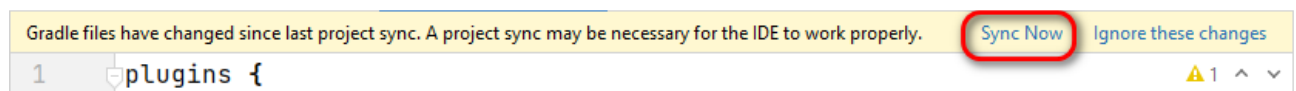
```
defaultConfig {
    applicationId = "com.example.exercise01"
    minSdk = 24
    targetSdk = 33
    ...
}
```

- To samo ustawiało się we wcześniejszych wersjach AS w pliku *AndroidManifest.xml* w węźle *uses-sdk*: (to widać w pliku projektu *AndroidManifest.xml* – w trybie Merget Manifest):

```
<uses-sdk
    android:minSdkVersion="24"
    android:targetSdkVersion="33" />
```

- Uwaga: w Android Studio pierwszeństwo ma wpis w skrypcie gradle !
- W przypadku zmian w plikach Gradle zazwyczaj wymagana jest synchronizacja – trzeba wtedy wybrać opcję *Sync Now* w pojawiającym się pasku podpowiedzi okna pliku lub w opcji menu:

Tools → *Android* → *Sync Project with Gradle files*



- Dużo podstawowych opcji dla projektu (wersje SDK, Gradle, wersje pakietów/bibliotek itp.) można ustawić wybierając opcję platformy:

File → *Project Structure...* – przeglądaj dostępne opcje.

- Plik *AndroidManifest.xml* zawiera opis składowych i atrybuty dla całej aplikacji, w tym:
 - `<application>` - węzeł dla całej aplikacji opisujący jej składowe i atrybuty – np. atrybut *icon* zawierający referencję do zasobu ikony uruchomienia aplikacji, atrybut *theme* zawierający referencję do zasobu opisującego motyw aplikacji.
 - `<activity>` - dla każdej aktywności węzeł opisujący aktywność i jej atrybuty

2.3. Konfigurowanie interfejsu i zasobów interfejsu

2.3.1. Zmiana układu okna ekranu aktywności

Domyślnym układem graficznym (*Layout*) nowo tworzonej aktywności jest *ConstraintLayout*. Tu zmieniamy układ aktywności głównej na linearny (*LinearLayout*). Można to zrobić dwójako:

- Edytując kod w trybie tekstowym:

Otwieramy **okno edycji pliku interfejsu (ekranu) aktywności** - w węźle *app\res\layout* plik **activity_main.xml**, wybieramy widok tekstowy (zakładka *Text*) i zamieniamy nazwę węzła *ConstraintLayout* (łącznie ze specyfikacją pakietu) na *LinearLayout* oraz dopisujemy w nim atrybut *android:orientation* równy *vertical*:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    ...
</LinearLayout>
```

W tym przypadku powinno się usunąć atrybuty komponentów używane tylko w domyślnym układzie *ConstraintLayout*.

- Narzędziem *Design*:

Otwieramy **okno edycji pliku interfejsu (ekranu) aktywności** - w węźle *app\res\layout* plik **activity_main.xml**, wybieramy widok projektowania (opcja *Design*). W okienku *Component Tree* zaznaczamy główny węzeł *ConstraintLayout* i z menu kontekstowego (prawy klawisz myszy) wybieramy opcję *Convert view ...*

Po konwersji jeszcze raz zaznaczamy układ w *Component Tree* jeśli chcemy zmienić orientację (na *horizontal* lub *vertical*).

2.3.2. Odwołania do zasobów, zasoby i identyfikatory, atrybuty komponentów widoków

Zdefiniowanie zasobu lub zmianę odwołania (identyfikator) do zasobów dla wyświetlanego komponentu ekranu (**widoku**) – np. tekstu na przycisku – można to zrobić na kilka sposobów: zmiany ręcznie w plikach xml, zmiany w oknie własności obiektów, opcje faktoryzacji kodu itp.

Zasoby tekstowe są zawarte w pliku **strings.xml** w węźle *app\res\values*,

Tu definiujemy i zmieniamy tekst etykiety *TextView* – zmiana z "Hello World!" na swój własny.

- W pliku **strings.xml** dodaj zasób: napis "Hello user!" o identyfikatorze *txt1* - ręcznie dopisz:

```
<string name="txt1">My 1st activity!</string>
```

Użyciem do tego samego celu edytora zasobów *Resource Manager* jest opisane w dalszej części.

- Zmień odwołanie atrybutu *text* dla kontrolki tekstowej na odniesienie do zasobu *txt1* w **strings.xml**, w pliku xml układu aktywności głównej **activity_main.xml**:

```
<TextView
    ...
    android:text="@string/txt1" />
```

- Zwróć uwagę na zmianę w oknie *Design* interfejsu aktywności w oknie *Properties* (po prawej stronie) własność *text*.

Dodawanie i zmiany i atrybutów widoków można zrobić w oknie *Design* oraz edytując kod xml.

- Zmień właściwości widoku tekstowego.

W trybie *Design* wybierz w oknie *Properties* właściwość *layout_width* i nadaj jej wartość *match_parent* lub *fill_parent*, lub *0dp* (wszystkie opcje robią to samo).

- Zwróć uwagę na zmianę w oknie tekstowym opisu interfejsu.
- Dodaj w trybie tekstowym atrybut *textAlignment* i ustaw wartość na *center*. Ostatecznie powinno się otrzymać:

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAlignment="center"
    android:text="@string/txt1" />
```

- Zmień w pliku *strings.xml* dla atrybutu *txt1* wartość napisu na taką z twoim imieniem i nazwiskiem.
- Sprawdź efekty zmian w plikach i oknach właściwości.

2.3.3. Dodawanie nowych elementów interfejsu

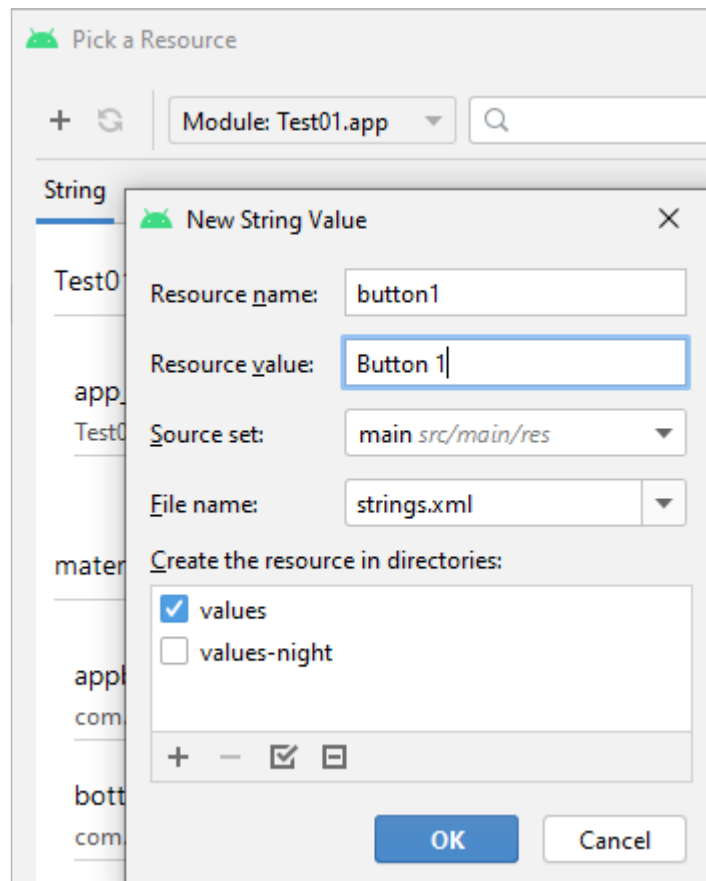
Dodawanie i edycja elementów ekranu aktywności możliwa jest zarówno w trybie *Design* jak i bezpośrednio w kodzie plików xml.

- Dodaj dwa przyciski - obiekty typu *Button*.
 - Pierwszy posługując się oknem interfejsu w trybie *Design* – przeciągając obiekt z okna *Palette*.
 - Drugi dodając w trybie tekstowym węzeł:

```
< Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:id="@+id/button2"
/>
```

Ewentualnie można skopiować w pliku xml istniejący element i kopię edytować (nadając nowe wartości atrybutów – np. unikalny ID).

- Utwórz i ustaw odwołania do następujących zasobów:
 - dla pierwszego przycisku do tekstu w *strings.xml*
 - w oknie *Design* interfejsu zaznacz pierwszy przycisk
 - w oknie *Properties* dla atrybutu *text* kliknij w element *[]*
 - w oknie *Pick a Resource* wybierz ikonę + (plus) = *Add new resource* → *String value...*
 - wpisz nazwę *button1* i wartość tekstu: *"Button 1"*



- w oknie tekstowym xml układu jeśli nie ma identyfikatora kontrolki to dodaj go (podobnie jak dla drugiego przycisku) – dopisz:

```
android:id="@+id/button1"
```

- sprawdź zawartość pliku strings.xml
- dla drugiego przycisku do tekstu w strings.xml
- dodaj w oknie tekstowym xml układu atrybut dla drugiego przycisku:

```
<Button
    android:id="@+id/button2"
    ...
    android:text="@string/button2"
    ... />
```

- dodaj w strings.xml węzeł:

```
<string name="button2">Button 2</string>
```

- Wypośrodkuj pierwszy przycisk używając w oknie *Properties* przycisku właściwości *layout:gravity* równej *center_horizontal*.
- Zmień rozmiar czcionki dla napisu powitalnego w trybie *Design* i zobacz efekt w kodzie xml.
- Dodaj w dowolny sposób kolejne dwa przyciski o identyfikatorach *button3* i *button4*.

Test działania

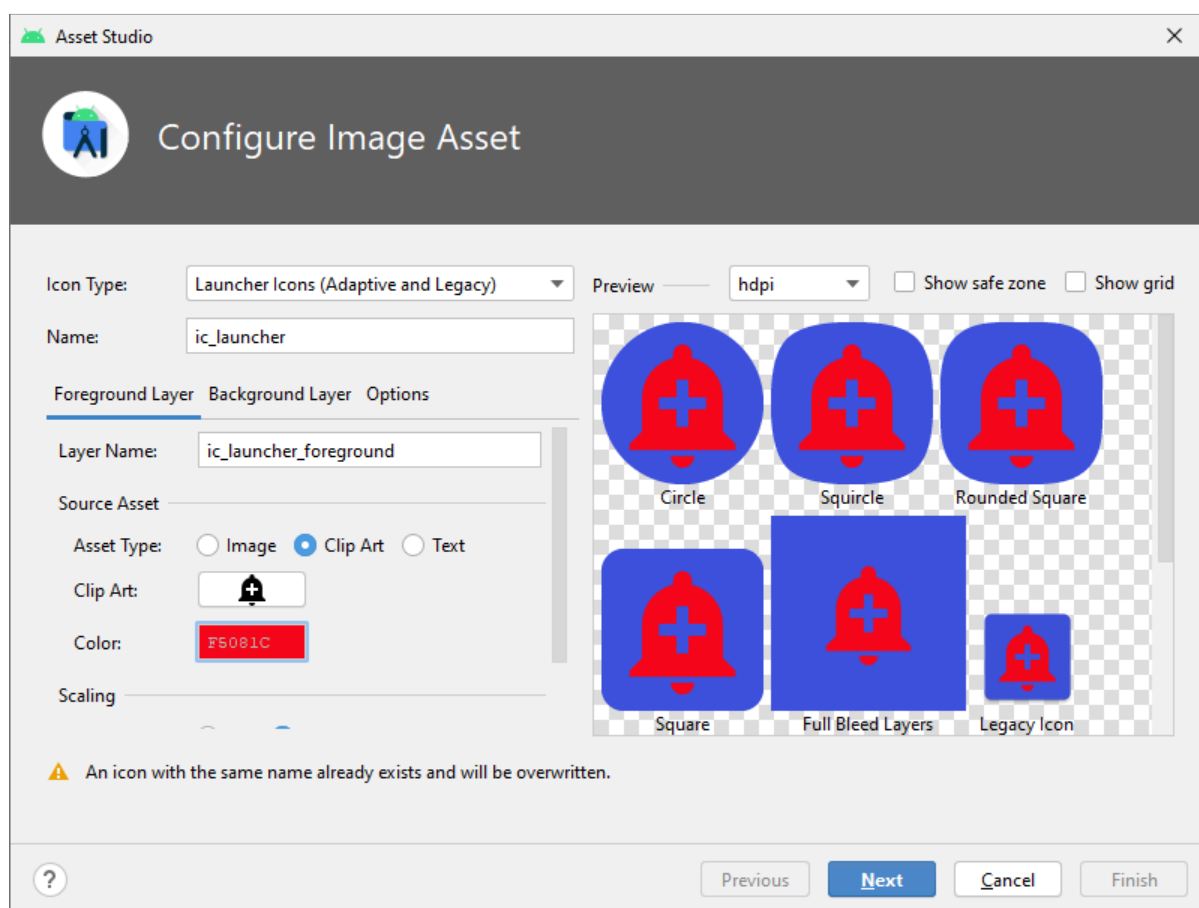
- Uruchom aplikację bezpośrednio na podpiętym smartfonie (lub gdy brak to na emulatorze) i sprawdź działanie.

2.4. Elementy graficzne aplikacji typu obrazki i ikony

Dodawanie obrazków używanych w aplikacji można zrobić za pomocą *Resource Manager* (tak samo jak w punkcie 2.3.3) – wybierając zakładkę okna *Resource Manager* (na lewym pasku AS obok zakładki Project) i w oknie zakładkę *Drawable* a następnie opcję *+/Import Drawable*. Te zasoby graficzne znajdują się w węźle *app\res\drawable*.

- Przygotuj niewielki obrazek w wybranym folderze i zaimportuj go do projektu. Wykorzystany zostanie w innej aktywności.

Zasoby graficzne można utworzyć również w węźle *app\res\mipmap*. Jedną z zasadniczych różnic jest generowanie przez platformę skalowanych wersji zasobu dla różnych rozdzielczości ekranu. Zazwyczaj są tu lokowane zasoby typu ikony używane w aplikacji. Ikony mogą być typu rastrowego lub wektorowego. Tworzenie wspierają narzędzia *Image Asset* i *Vector Asset*, dostępne w *Resource Manager* dodając zasób dla zakładki mipmap (rys.)



- Korzystając z gotowych ikon dostępnych *Image Asset* utwórz własną ikonę startową dla aplikacji.
- Ustaw w pliku manifestu odwołanie do utworzonych ikon dla atrybutów *icon* i *roundIcon* specyfikując wartości atrybutów w postaci:
`android:icon="@mipmap/nazwa_ikony"`

2.5. Tworzenie kolejnych aktywności

Kolejne ekrany aplikacji są implementowane jako oddzielne aktywności.

- Utwórz w aplikacji **drugą** aktywność (bez zmiany układu (layout'u)) – np. o nazwie **Activity2**:
 - zaznacz węzeł app w okienku projektu
 - wybierz opcję *File → New → Activity → Empty Views Activity* lub zrób to samo opcją z menu kontekstowego.
- Wstaw u góry ekranu napis identyfikujący – np.: "Activity 2".
- Skonfiguruj odpowiednie zasoby (tekst) i odwołania podobnie jak w pierwszej aktywności.
- Przygotuj w podobny sposób **trzecią** aktywność (np. o nazwie **Activity3**)

2.6. Dodanie funkcjonalności nawigacji między aktywnościami, obsługa zdarzeń

Wywołanie innej aktywności można wykonać na kilka sposobów. Tu przedstawiane są sposoby z bezpośrednim definiowaniem **słuchaczy zdarzeń** powiązanymi z elementami ekranu realizowane na różne sposoby.

Tu wywołanie aktywności realizuje się z użyciem intencji (obiektów typu **Intent**) i metody aktywności **startActivity(...)**. Wywoływanej na skutek zdarzenia *Click* kliknięcia obiektu.

2.6.1. Słuchacz w postaci klasy anonimowej:

- Otwórz plik java/kotlin **pierwszej** aktywności.

Szablony projektu innego niż zalecany w ćwiczeniu mogą zawierać wygenerowane różne metody do implementacji (np. dla menu). W takim przypadku **pozostaw te metody nietknięte**.
- W metodzie onCreate dopisz kod:
 - uzyskania referencji do przycisku 1
 - tworzenia intencji dla wywołania drugiej aktywności
 - utworzenia słuchacza dla zdarzenia kliknięcia przycisku 1 (poprzez klasę anonimową)
 - przypisania słuchacza do przycisku metodą **setOnClickListener**
 - wywołania drugiej aktywności w słuchaczu (po odebraniu zdarzenia *Click*)

```
val button1: Button = findViewById(R.id.button1)
button1.setOnClickListener { _ ->
    val myIntent = Intent(this, Activity2::class.java)
    startActivity(myIntent)
}
```

Powyżej do definicji słuchacza zostało użyte wyrażenie lambda.

2.6.2. Słuchacz implementowany przez klasę aktywności

- Rozszerzamy klasę aktywności o interfejs **OnClickListener** :

```
class MainActivity : AppCompatActivity(), View.OnClickListener {
    ...
}
```

- Dopisujemy w klasie (ewentualnie kompilator zaproponuje wygenerowanie) wymaganą metodę interfejsu słuchacza **onClick**, w której implementujemy wywołanie 3-ciej aktywności identycznie jak w klasie anonimowej.

```
override fun onClick(view: View?) {
    val myIntent = Intent(this, Activity3::class.java)
    startActivity(myIntent)
}
```

- Dopisujemy w metodzie **onCreate** przypisanie słuchacza do przycisku 2:

```
override fun onCreate(savedInstanceState: Bundle?) {
    ...
    val button2: Button = findViewById(R.id.button2)
    button2.setOnClickListener(this)
}
```

2.6.3. Definiowanie handler'a dla definiowanego atrybutu zdarzenia dla kontrolki

Ta metoda definiuje atrybut zdarzenia dla konkretnej kontrolki.

Uwaga: w niektórych sytuacjach nie wolno jej używać ponieważ może prowadzić do nieprzewidywalnych awarii aplikacji (np. podczas używania tzw. fragmentów).

- Otwórz plik interfejsu (layout) aktywności g1 (głównej).
- Dopisz w trybie tekstowym atrybut deklarujący handler do przycisku 3:

```
android:onClick="run2"
```

- Dopisz metodę **run2** w pliku java/kotlin aktywności 1:

```
fun run2() { //or: run2(view: View) - but view is not used here
    val myIntent = Intent(this, Activity2::class.java)
    startActivity(myIntent)
}
```

2.6.4. Definiowanie oddzielnej (wewnętrznej) klasy/zmiennej słuchacza

Ta metoda jest o tyle uniwersalna, że pozwala oddzielnie w jednym miejscu definiować kod obsługi danego zdarzenia dla wszystkich komponentów/kontrolki aktywności.

- Zdefiniuj w klasie głównej aktywności zmienną słuchacza typu **OnClickListener** zdarzeń *Click*, w którym będzie sprawdzany identyfikator obiektu dla którego zaszło zdarzenie i uruchomiona zostanie 3-cia aktywność– dopisz w klasie kod dla zdarzeń od przycisku 4:

```
val myListener = View.OnClickListener { view ->
    when (view.getId()) { //better prop. syntax: when(view.id)
        R.id.button4 -> {
            val myIntent = Intent(this, Activity3::class.java)
            startActivity(myIntent)
        }
    }
}
```

- w metodzie **onCreate** przypisanie słuchacza do przycisku 4:

```

override fun onCreate(savedInstanceState: Bundle?) {
    ...
    val button4: Button = findViewById(R.id.button4)
    button4.setOnClickListener(myListener)
}

```

Test działania

- Uruchom aplikację bezpośrednio na smartfonie lub emulatorze i sprawdź działanie.

2.7. Metody cyklu życia i obsługa powrotu do poprzedniej aktywności

Podstawowe metody aktywności związane z jej cyklem życia to:

onCreate, onStart, onResume, onRestart, onPause, onStop, onDestroyed.

Wstawiając tam odpowiedni kod można uzyskać działanie w odpowiednich momentach życia aktywności.

- W trzeciej aktywności w pliku java/kotlin dopisz metodę **onStart** a w niej kod tworzący powiadomienie (**Toast**) wyświetlające napis "Activity 3 is started"

```

override fun onStart() {
    super.onStart()
    val toast: Toast = Toast.makeText(this,
                                     "Activity 3 is started",
                                     Toast.LENGTH_LONG)
    //toast.setGravity(Gravity.CENTER, 0, 0)
    toast.show()
}

```

Tu używamy obiektu typu **Toast** dla wyświetlenia przez chwilę krótkiego komunikatu.

Atrybut pozycjonowania na ekranie **Gravity** niestety dla nowych API nie działa ☹.

Zakończenie/powrót z aktywności można uzyskać stosując metody aktywności **onBackPressed()** i **finish()**.

- W trzeciej aktywności w pliku java/kotlin dopisz w metodzie **onCreate** kod, który tworzy słuchacza na długie dotknięcie (*LongClick*) całego okna i po wystąpieniu takiego zdarzenia ten słuchacz zamknie aktywność – słuchacza ustawiamy dla całego okna.

```

class Activity3 : AppCompatActivity(), OnLongClickListener {
    ...
}

```

- Dopisujemy w klasie (ewentualnie kompilator zaproponuje wygenerowanie) wymaganą metodę interfejsu słuchacza **onLongClick** w której możemy wywołać metodę **onBackPressed** lub **finish**, które powodują powrót do wcześniejszej aktywności zakończenie aktualnej aktywności (i również powrót do poprzedniej):

```

override fun onLongClick(view: View?): Boolean {
    onBackPressed()
    return true
}

```

- W metodzie **onCreate** ustawimy słuchacza implementowanego przez aktywność:

```
val activity3: View = findViewById(R.id.activity3)
activity3.setOnLongClickListener(this)
```

activity3 to identyfikator układu (layout'u) drugiej aktywności, który ewentualnie trzeba dodać (np. w oknie *Properties* po zaznaczeniu obiektu w trybie Design).

- Wstaw na środku ekranu tekst: "Press me long...".
- Dodaj na ekranie przycisk - obiekt typu Button - na dole ekranu.
 - Ustaw mu napis "Powrót",
- Dopisz w handler do przycisku:

```
android:onClick="finishActivity3"
```

- Dopisz metodę finishActivity3(...) w pliku java/kotlin aktywności 3, w której będzie kod wywołujący zakończenie aktywności:

```
fun finishActivity3() { //or: finishActivity(view: View) but ...
    finish()
}
```

Powrót do wybranej aktywności można również zdefiniować w pliku manifestu.

- W pliku manifestu dopisz dla aktywności 2 i 3 atrybut **parentActivityName**, który spowoduje dodanie na pasku ekranu aktywności (*Activity Bar*) elementu (ikony) powrotu do głównej aktywności:

```
<application>
...
<activity
    android:name=".Activity3"
    android:parentActivityName=".MainActivity"
...
</application>
```

Test działania

- Uruchom aplikację bezpośrednio na smartfonie lub emulatorze i sprawdź działanie.

2.8. Odziaływanie zdarzeniami na elementy ekranu (interfejsu)

Tu będzie operowanie na obrazku za pomocą przełącznika dwustanowego (switch'a)

- Wstaw do układu drugiej aktywności (przeciągając je z okienka *Palette*) dwa komponenty:
 - obrazek **<ImageView>**
 - kontrolkę **<Switch>**

Ustaw odpowiednio położenie elementów.

Nadaj im odpowiednie identyfikatory (np. **img1** i **switch1**)

- Dla obrazka ustaw referencję na zasób obrazka, który wcześniej dodano do projektu – np.:
 "@drawable/image_name"
 gdzie *image_name* to nazwa pliku obrazka.
 Wybór zasobu/obrazka może pojawić się przy wstawianiu elementu.
- Dodaj w klasie zmienną globalną, która będzie zainicjowana później:

```
lateinit var img1: ImageView
```

- W metodzie onCreate aktywności pobierz referencje do obiektów i ustaw słuchacza **OnCheckedChangeListener** dla przełącznika:

```
img1 = findViewById(R.id.img1)
val sw: Switch = findViewById(R.id.switch1)
sw.setChecked(true)
sw.setOnCheckedChangeListener(swListener)
```

- Dopisz w klasie aktywności słuchacza **swListener**, w którym będzie ukrywany lub pokazywany obrazek (ustawiany atrybut widoczności) w zależności od stanu przełącznika.

```
val swListener = CompoundButton.OnCheckedChangeListener { _, isChecked ->
    if (isChecked)
        img1.setVisibility(View.VISIBLE)
    else
        img1.setVisibility(View.INVISIBLE)
}
```

Zwróć uwagę, że ten słuchacz ma dwa parametry przy czym pierwszy jest pomijany z użyciem w kotlinie symbolu podkreślenia `_`.

2.9.

*TU PLANOWANE JEST DODANIE JESZCZE JEDNEGO ELEMENTU PROGRAMOWANIA APLIKACJI
(TAK SZYBKO JAK SIĘ DA)*

2.10. Test działania aplikacji

- Uruchom aplikację bezpośrednio na smartfonie lub emulatorze i sprawdź działanie.
- Zwróć uwagę na wszystkie elementy interfejsu (również na pasku tytułowym i przetestuj wszystkie zaimplementowane funkcje).

3. Zadanie – część II.

A. Wykonać na zajęciach elementy aplikacji wg wymagań i wskazówek przekazanych przez prowadzącego.

lub

B. Program przygotowawczy do samodzielnego wykonania "w domu":

Napisać aplikację składającą się z kilku aktywności, spełniającą wymagania wyspecyfikowane przez prowadzącego.

Wymagane umiejętności – elementy programistyczne do opanowania/realizacji.

1. Tworzenie podstawowych aktywności oraz aktywności ze wsparciem dla starszych systemów (w trybie kompatybilności).
2. Definiowanie i konfiguracja różnego typu układów graficznych interfejsu graficznego (*Layouts*) w tym: liniowego, relatywnego, *table*, *constraint*), oraz funkcji przewijania (*ScrollView*).
3. Umieszczanie i różne pozycjonowanie kontrolek interfejsu w układach graficznych.
4. Konfigurowanie i obsługa zdarzeń podstawowych kontrolek interfejsu w tym: etykiety tekstowe, przycisk zwykły (*button*), przycisk dwustanowy, tekstowe pole edycyjne, przyciski wyboru jednokrotnego (*radio buttons*) i wielokrotnego (*checkbox*), obrazek.
5. Obsługa reakcji na zdarzenia kliknięcia i tzw. "długiego" kliknięcia (*long click*).
Dwa sposoby programowania obsługi – definiowanie słuchacza w kodzie java/kotlin i handler'a w pliku zasobów .xml i kodzie java/kotlin.
6. Tworzenie aplikacji składającej się z kilku aktywności. Uruchamianie aktywności z użyciem intencji jawnych (zatem również tworzenie i inicjowanie intencji jawnych).
7. Obsługa powrotu do poprzedniej aktywności (lub wyjścia z aplikacji) za pomocą akcji (np. kliknięcia przycisku) generowanej w aplikacji i definiowaniu aktywności nadrzędnej w manifestie, oraz zamykanie aktywności.
8. Wyświetlanie prostych wyskakujących powiadomień na ekranie typu *pop-up* (*Toast*).
9. Definiowanie podstawowych właściwości aplikacji w plikach *AndroidManifest* i *Styles*, tj.: ikona, etykieta, motyw (*theme*).
10. Podstawowa wiedza o metodach *callbacks* wywoływanych przy przejściach pomiędzy stanami aplikacji.

Uwaga: na zaliczenie wymagana jest znajomość kodu i umiejętność modyfikacji aplikacji dla osiągnięcia żądanych zmian.