





## Wykład 2 Podstawowe elementy aplikacji Android

### Programowanie Urządzeń Mobilnych

Dr inż. Damian Raczyński

## Struktura aplikacji

Aplikacja składa się z ze zbioru danych i kodu (wielu elementów)

- Elementy te są przechowywane w prywatnym obszarze w systemie plików, 
- Uruchamiane w kontekście użytkownika, który został utworzony w procesie instalacji, 
- Aplikacje są uruchamiane w ich własnych VM – izolacja aplikacji od siebie, 
- Aplikacja musi zgłaszać potrzebę korzystania z danych urządzenia. 

## Struktura aplikacji

**Aplikacja Andorid nie posiada pojedynczego punktu startowego (jak w innych systemach).**

Każda aplikacja składa się z jednego lub wielu komponentów, które są uaktywniane **ZGODNIE Z WOLĄ SYSTEMU OPERACYJNEGO LUB UŻYTKOWNIKA (PUNKTY WEJŚCIA DO APLIKACJI)**.

(tworzenie, startowanie, budzenie ...)

Komponenty tej samej aplikacji są udostępniane w pojedynczym pakiecie (apk – Android Package) i są zdefiniowane w pliku manifest.

(apk – plik zip z hierarchią aplikacji)

3

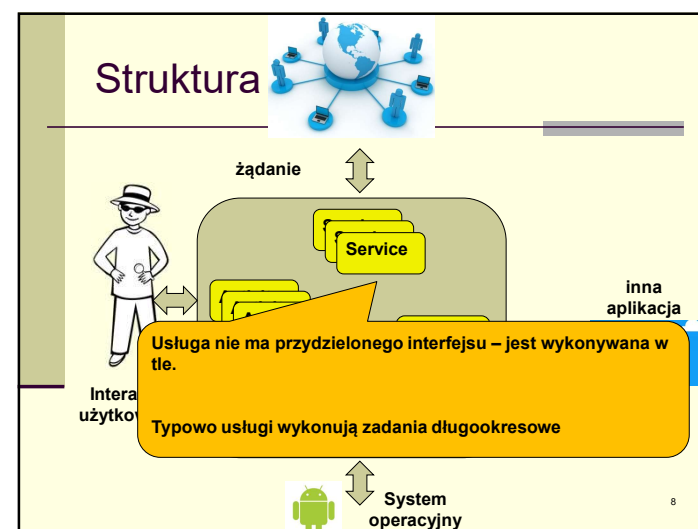
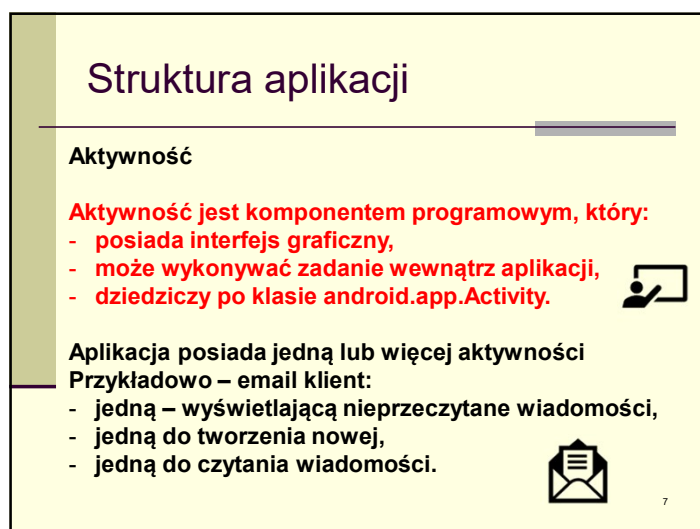
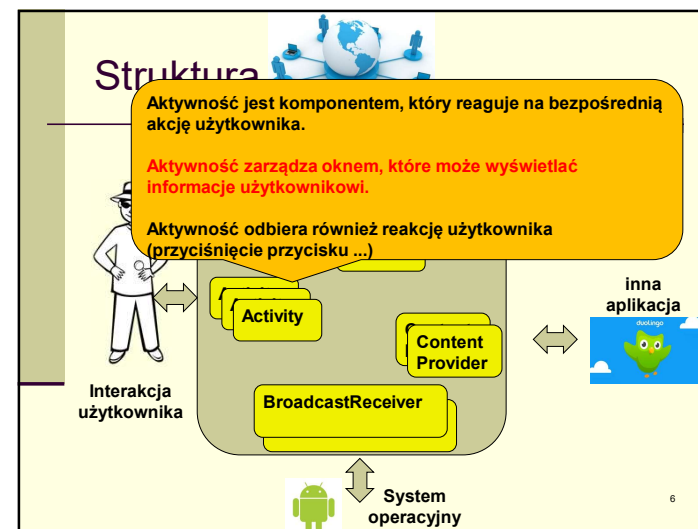
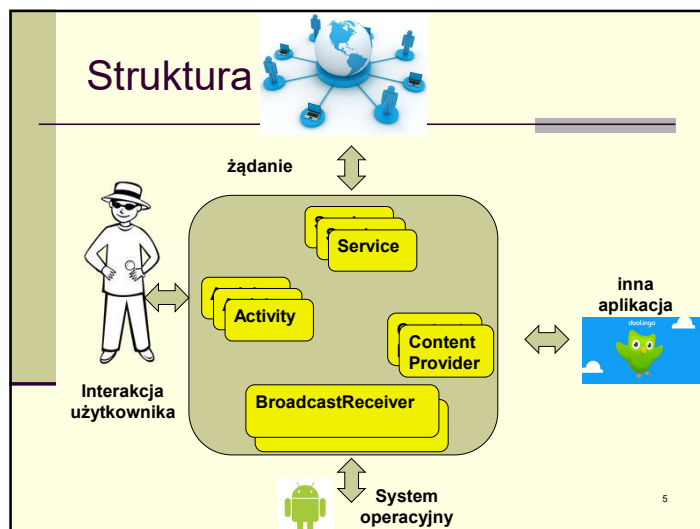
## Struktura aplikacji

**Komponenty (jeden lub więcej na aplikację):**

- **Aktywności (Activity),**
- **Usługi (Service),**
- **Dostawcy Treści (ContentProvider),**
- **BroadcastReceiver.**

Każdy z wymienionych elementów odpowiada za określaną interakcję wewnątrz systemu operacyjnego.

4



## Struktura aplikacji

### Usługa

Każdy inny komponent aplikacji może uruchomić usługę (np. aktywność – okienko FTP, usługa – proces pobierania pliku).

Zazwyczaj usługi wykorzystywane są do zadań długotrwałych

- Usługa może odtwarzać muzykę, gdy użytkownik obsługuje inną aplikację,
- Usługa może pobierać dane z sieci bez blokowania interakcji użytkownika z inną aktywnością,

Usługi są podklasami klasy `android.app.Service`

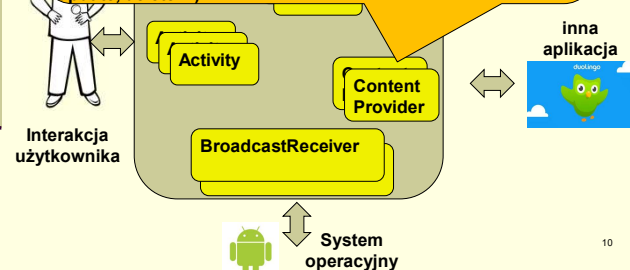
9

## Struktura

Komponent, który może być potrzebny, gdy aplikacja zarządza danymi.

Element dostarcza danych dla innych aplikacji (np. lista znanych przechowywana wewnątrz aplikacji).

Komponent musi zapewnić odpowiedni interfejs (insert, update, delete ...)



10

## Struktura aplikacji

### ContentProvider

Dostarcza danych np. z systemu plików, bazy SQLite, sieci web, ...

Implementuje zbiór standardowych metod, które umożliwiają innym aplikacjom pobieranie lub modyfikowanie danych obsługiwanych przez aplikację.

ContentProvider jest podklasą klasy `android.content.ContentProvider`

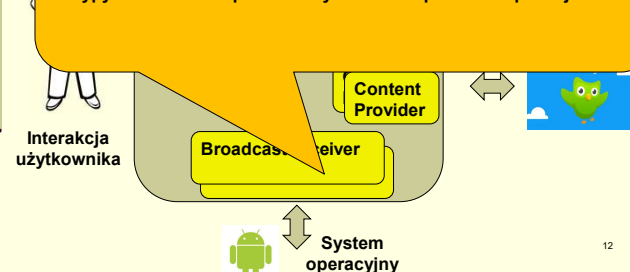
11

## Struktura

Komponenty rejestrowane w celu reagowania na specjalne zdarzenia.

np. BroadcastReceiver reagujące na niski poziom baterii/wyłączenie ekranu/wykonano zdjęcie.

Typy: zdarzenia na poziomie systemu / na poziomie aplikacji



12

## Struktura aplikacji

### BroadcastReceiver

**BroadcastReceiver jest komponentem, który czeka na wiadomości.**  
**Pewne wiadomości są tworzone przez system operacyjny (np. wyłączenie ekranu, niski poziom baterii, ...)**

Aplikacje mogą również generować wiadomości (np. gdy transfer danych został zakończony)

BroadcastReceiver nie posiada interfejsu graficznego, ale może generować wiadomości w pasku stanu.

13

## Plik manifestu

Funkcjonalność dostarczona przez aplikację musi zostać zdefiniowana w pliku manifestu

Jest to dokument XML zawierający „kontrakt” pomiędzy aplikacją a systemem operacyjnym

Zawiera listę poszczególnych komponentów, zawartych w aplikacji,

listę zezwoleń, których potrzebuje aplikacja (np. korzystanie z internetu, odczyt kontaktów ...)

Definiuje minimalną wersję API, którą wymaga aplikacja

Określa z jakiego sprzętu będzie korzystała aplikacja (aparat, gps, ...)

Określa biblioteki, z jakich aplikacja korzysta (np. Google maps).



14

## Plik manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
            android:label="@string/example_label" ... >
        </activity>
        <service> ... </service>
        <receiver>...</receiver>
        <provider>...</provider>
        <uses-feature android:name="android.hardware.camera.any"
            android:required="true" />
        <uses-sdk android:minSdkVersion="7"
            android:targetSdkVersion="19" />
        ...
    </application>
</manifest>
```

## Plik manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
            android:label="@string/example_label" ... >
        </activity>
        <service> ... </service>
        <receiver>...</receiver>
        <provider>...</provider>
        <uses-feature android:name="android.hardware.camera.any"
            android:required="true" />
        <uses-sdk android:minSdkVersion="7"
            android:targetSdkVersion="19" />
        ...
    </application>
</manifest>
```

Ustawienia aplikacji,  
 Android:icon="@drawable..." – ikona  
 aplikacji z zasobów

## Plik manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    <activity android:name="com.example.project.ExampleActivity"
      android:label="@string/example_label" ... >
      </activity>
    <service> ... </service>
    <receiver>...</receiver>
    <provider>...</provider>
    <uses-feature android:name="android.hardware.camera.any"
      android:required="true" />
    <uses-sdk android:minSdkVersion="7"
      android:targetSdkVersion="19" />
    ...
  </application>
</manifest>
```

**<activity>** – definiuje aktywności aplikacji,  
**android:name** – pełna kwalifikowana nazwa klasy aktywności,  
**android:label** – nazwa widziana przez użytkownika

## Plik manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    <activity android:name="com.example.project.ExampleActivity"
      android:label="@string/example_label" ... >
      </activity>
    <service> ... </service>
    <receiver>...</receiver>
    <provider>...</provider>
    <uses-feature android:name="android.hardware.camera.any"
      android:required="true" />
    <uses-sdk android:minSdkVersion="7"
      android:targetSdkVersion="19" />
    ...
  </application>
</manifest>
```

Pozostałe 3 rodzaje komponentów aplikacji

## Plik manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    <activity android:name="com.example.project.ExampleActivity"
      android:label="@string/example_label" ... >
      </activity>
    <service> ... </service>
    <receiver>...</receiver>
    <provider>...</provider>
    <uses-feature android:name="android.hardware.camera.any"
      android:required="true" />
    <uses-sdk android:minSdkVersion="7"
      android:targetSdkVersion="19" />
    ...
  </application>
</manifest>
```

Wymagania sprzętowe aplikacji

## Niezbędne minimum



Do określania jakie możliwości systemu (sprzęt i funkcje) są niezbędne lub zalecane do prawidłowego działania aplikacji służy znacznik **<uses-feature>** (wylącznie cele informacyjne, ale np. [Google Play](#) po nich filtruje)

```
<uses-feature
  android:name="android.hardware.sensor.light"/>
<uses-feature
  android:name="android.hardware.sensor.proximity" />
```

Do działania aplikacji wymagany jest zarówno czujnik światła jak i czujnik zbliżenia.

## Niezbędne minimum



```
<uses-feature
  android:name="string"
  android:required=["true" | "false"]
  android:glEsVersion="integer" />
```

**android:name** – określa pojedynczą cechę sprzętu/oprogramowania wykorzystywaną przez aplikację

**android:required** – "true" dla funkcji bez której aplikacja nie będzie działała prawidłowo (domyślnie true)

**android:glEsVersion** – wersja OpenGL ES potrzebna dla aplikacji

21

## Plik manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    <activity android:name="com.example.project.ExampleActivity"
      android:label="@string/example_label" ... >
      </activity>
    <service> ... </service>
    <receiver>...</receiver>
    <provider>...</provider>
    <uses-feature android:name="android.hardware.camera.any"
      android:required="true" />
    <uses-sdk android:minSdkVersion="7"
      android:targetSdkVersion="19" />
    ...
  </application>
</manifest>
```

Minimalna wersja SDK konieczna do poprawnego działania aplikacji i wersja zalecana

## Niezbędne minimum



### Elementy pliku manifest związane z wymaganiami aplikacji

Element	Opis
uses-permission	Definiuje pozwolenie, które użytkownik MUSI przydzielić aplikacji, aby działała poprawnie (przy instalacji).
uses-configuration	Definiuje wymagania związane z bibliotekami/sprzętem. Element wykorzystywany aby uniknąć instalacji aplikacji, które nie będą działały na danym urządzeniu
uses-library	Określa wymagania związane z bibliotekami. W przypadku braku odpowiednich bibliotek aplikacja nie będzie wyświetlana w Google Play

23

## Plik manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    ...
    <uses-permission
      android:name="android.permission.WRITE_EXTERNAL_STORAGE"
      android:maxSdkVersion="20" />
    <uses-configuration android:reqHardKeyboard="true"
      android:reqTouchScreen="finger" />
    <uses-library android:name="com.google.android.maps"/>
    <compatible-screens>
      <screen android:screenSize="small"
        android:screenDensity="ldpi" />
      <screen android:screenSize="normal"
        android:screenDensity="ldpi" />
    </compatible-screens>
    ...
  </application>
</manifest>
```

## Plik manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    ...
    <uses-permission
      android:name="android.permission.WRITE_EXTERNAL_STORAGE"
      android:maxSdkVersion="20" />
    <uses-configuration android:reqHardKeyboard="true"
      android:reqTouchScreen="finger" />
    <uses-library android:name="com.google.android.maps" />
    <compatible-screens>
      <screen android:screenSize="small"
        android:screenDensity="ldpi" />
      <screen android:screenSize="normal"
        android:screenDensity="mdpi" />
    </compatible-screens>
    ...
  </application>
</manifest>
```

Określa pozwolenia, których potrzebuje aplikacja do swojego działania. Pozwolenia są udzielane przez użytkownika w trakcie instalacji (powyżej Android 6.0 przy działającej aplikacji).

android:maxSdkVersion – od pewnej wersji SDK pozwolenia mogą nie być wymagane (np. WRITE\_EXTERNAL\_STORAGE od Api 19)

## Plik manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    ...
    <uses-permission
      android:name="android.permission.WRITE_EXTERNAL_STORAGE"
      android:maxSdkVersion="20" />
    <uses-configuration android:reqHardKeyboard="true"
      android:reqTouchScreen="finger" />
    <uses-library android:name="com.google.android.maps" />
    <compatible-screens>
      <screen android:screenSize="small"
        android:screenDensity="ldpi" />
      <screen android:screenSize="normal"
        android:screenDensity="mdpi" />
    </compatible-screens>
    ...
  </application>
</manifest>
```

Określa jakiego sprzętu lub oprogramowania aplikacja potrzebuje. Zapobiega to instalacji aplikacji na urządzeniu, na którym aplikacja nie będzie działała poprawnie.

## Plik manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    ...
    <uses-permission
      android:name="android.permission.WRITE_EXTERNAL_STORAGE"
      android:maxSdkVersion="20" />
    <uses-configuration android:reqHardKeyboard="true"
      android:reqTouchScreen="finger" />
    <uses-library android:name="com.google.android.maps" />
    <compatible-screens>
      <screen android:screenSize="small"
        android:screenDensity="ldpi" />
      <screen android:screenSize="normal"
        android:screenDensity="mdpi" />
    </compatible-screens>
    ...
  </application>
</manifest>
```

Określa jakiej biblioteki potrzebuje aplikacja do poprawnego działania

## Plik manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    ...
    <uses-permission
      android:name="android.permission.WRITE_EXTERNAL_STORAGE"
      android:maxSdkVersion="20" />
    <uses-configuration android:reqHardKeyboard="true"
      android:reqTouchScreen="finger" />
    <uses-library android:name="com.google.android.maps" />
    <compatible-screens>
      <screen android:screenSize="small"
        android:screenDensity="ldpi" />
      <screen android:screenSize="normal"
        android:screenDensity="mdpi" />
    </compatible-screens>
    ...
  </application>
</manifest>
```

Określa wszystkie typy ekranów obsługiwane przez aplikację

## Obiekt aplikacji

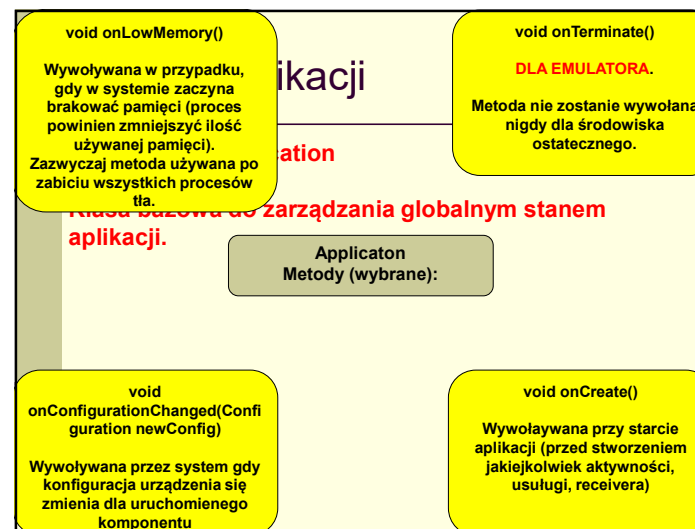
Dla każdej uruchamianej aplikacji, Android tworzy instancję klasy `android.app.Application`.

Możliwe jest wskazanie podklasy w pliku manifest, która będzie wykorzystywana do przechowywania danych współdzielonych pomiędzy komponentami.

Metody wywoływane w zależności od statusu obiektu aplikacji:

```
void onCreate()
void onConfigurationChanged(..)
void onLowMemory()
void onTerminate()
```

29



## Obiekt aplikacji

Tworzymy

com.example.root.myapplication  
App  
MainActivity

```
public class App extends Application {
    @Override
    public void onCreate() {
        Context context = getApplicationContext();
        CharSequence text = "Witaj świecie";
        int duration = Toast.LENGTH_SHORT;
        Toast toast = Toast.makeText(context, text, duration);
        toast.show(); //okienko z komunikatem Witaj świecie
        super.onCreate();
    }
}

<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    <application
        android:name="com.example.root.myapplication.App"
        ...>
    </application>
</manifest>
```

## Obiekt aplikacji

```
public class App extends Application {
    ...
    @Override
    public void onConfigurationChanged(Configuration n)
    {
        Context context = getApplicationContext();
        CharSequence text = String.valueOf(n.orientation);
        int duration = Toast.LENGTH_SHORT;
        Toast toast = Toast.makeText(context, text, duration);
        toast.show(); // 1 - dla wertykalnej, 2 - dla horyzontalnej
        super.onConfigurationChanged(n);
    }
}

<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    <application
        android:name="com.example.root.myapplication.App"
        ...>
    </application>
</manifest>
```



## Struktura aplikacji

System tworzy poszczególne komponenty składowe, po utworzeniu obiektu aplikacji  
- w zależności od zdarzeń, które się pojawiają.



Obiekt aplikacji znajduje się w pamięci dopóki istnieją aktywne komponenty.  
- obiekt aplikacji jest usuwany z pamięci, gdy wszystkie komponenty zakończyły ich cykl życia.

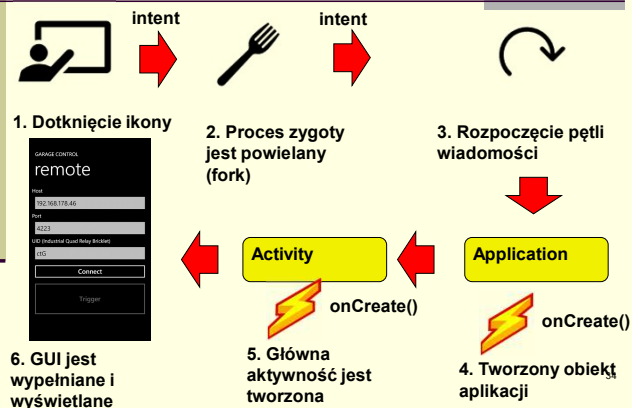


Możliwe jest utworzenie klasy pochodnej po Application  
- ona może odgrywać rolę Modelu we wzorcu MVC (używana do przechowywania i zarządzania danymi aplikacji)



33

## Uruchomienie aplikacji



## Uruchamianie aplikacji

Każdorazowo, gdy ikona ekranu głównego jest dotykana, nowy proces jest odpalany

Intencja jest swego rodzaju komunikatem, który podróżuje od aplikacji do aplikacji.

Intencji można użyć również do przemieszczania komunikatów pomiędzy jedną aplikacją (komunikacja aplikacji „z samą sobą”)

Co jest przesyłane – do jakich celów



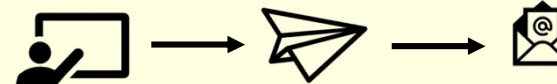
35

## Uruchamianie aplikacji

W zależności od potrzeby – android ustala adresata.



np. intencja – wysłać e-mail – powoduje utworzenie procesu klienta pocztowego (chyba, że już został utworzony – Android nie zabija procesów natychmiastowo).



36

## Uruchamianie aplikacji

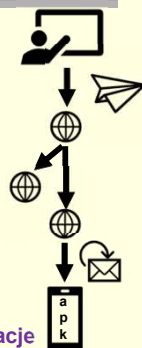
**Gdy pożądanego procesu nie było:**

Intencja jest dostarczana do procesu systemowego zygoty (zanim zostanie dostarczona do adresata, należy go utworzyć), który rozdziela się (forks).

Zygota zawiera wstępnie odpaloną VM – to redukuje czas uruchomienia aplikacji.

- Tylko określone klasy aplikacji muszą być załadowane, klasy „ogólnego przeznaczenia” już są załadowane w procesie rodzica.

- Rozdzielony proces (forked) rozpoczyna pętlę komunikatów i wykorzystuje informacje zawarte w intencji aby zlokalizować i załadować odpowiedni **APK** i plik **manifestu**.



37

## Uruchamianie aplikacji

Główny wątek rozdzielonego procesu tworzy obiekt **Application**

- wywoływana jest metoda **onCreate()**

Następnie aktywność określona przez otrzymaną intencję jest tworzona

- metoda **onCreate(...)** aktywności, większość kodu programisty – powinna się znaleźć tutaj (tworzenie interfejsu ...).

38

## Interakcja aplikacji

Interakcja między aplikacjami budowana jest poprzez wymianę komunikatów (aplikacje oddzielone od siebie w osobnych VM).



**SZCZEGÓLNY RODZAJ WIADOMOŚCI – INTENCJE.** mogą zostać użyte do: uruchamiania aktywności, uruchamiania usługi lub dostarczenia broadcastu.

Intencja określa akcję do wykonania i zbiór danych, na których wykonana zostanie operacja.

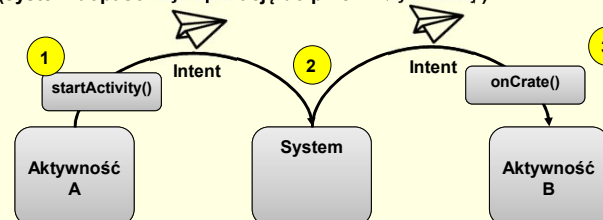
- System operacyjny odnajduje i wywołuje odpowiedni komponent/komponenty, który „umie” obsłużyć żądaną akcję (np., podajemy współrzędne – wyświetlana jest odpowiednia mapa ...).

## Intencje

Podział intencji:

- **Explicit intents** – określają konkretny komponent do wywołania (np. do wywołania komponentów własnej aplikacji). Wymagają podania kwalifikowanej nazwy klasy.

- **Implicit intents** – określają ogólną akcję do wykonania (system dopasowuje aplikację do przez filtry intencji):



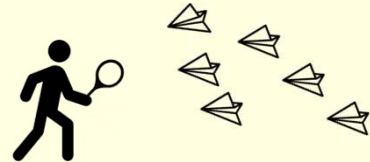
40

## Filtry intencji

Wyrażenie w pliku manifestu, które określa typy intencji, które komponent aplikacji jest w stanie obsłużyć.

W przypadku, gdy aktywność nie ma przypisanego filtra intencji jej wywołanie jest możliwe tylko za pomocą intencji typu **explicit**.

Od Android 5.0 system wygeneruje wyjątek, jeżeli usługę (`bindService`) wywoła się w sposób **implicit** (względny bezpieczeństwa).



41

## Plik manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    <activity ...>

      <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="http"
              android:host="www.pwsz.nysa.pl" />
      </intent-filter>

    </activity>
  </application>
</manifest>
```

## Intencje

Intencja jest abstrakcyjnym zapisem operacji, która ma zostać wykonana.

Może składać się z następujących elementów:

Intencja
action
data
Dodatkowe atrybuty
category
type
component
extras

43

## Intencje

**NAZWA KOMPONENTU**- tworzy intencję typu **explicit**. Zalecana do użycia w przypadku usług. Metody: `setComponent()`, `setClass()`, `setClassName()` lub konstruktor. Nazwy kwalifikowane w postaci: `com.example.ExampleService`.

**AKCJA** – unikalny string określający co należy zrobić (`android.intent.action.*` lub własne wartości). Rodzaj akcji określa jak interpretować pozostałe pola. `setAction()` lub konstruktor.

(`ACTION_VIEW`, `ACTION_DIAL`, `ACTION_EDIT`, ...)

44

## Intencje

**DANE** - dane, na których zostanie wykonana operacja zazwyczaj wyrażenie URI (Uniform Resource Identifier)  
`setData()`, `setType()` (typ danych), `setDataAndType()`  
 Do przesłania danych i typu należy wywołać ostatnią metodę (poprzednie czyszczą się nawzajem).

Typ danych	Znaczenie
scheme	Protokół, adres Uri (np. content, http, ftp, ...)
host	Nazwa hosta (www.google.pl).
port	Numer portu
path	Ścieżka (np. /contact/)
mimetype	Typ danych

45

## Intencje

**KATEGORIA** – jeden lub więcej stringów zawierających dodatkowe informacje na temat rodzaju komponentu, który powinien obsłużyć intencję (np. gdy daną akcję może obsłużyć wiele elementów – do wyboru odpowiednich).  
`addCategory()`

Kategoria	Znaczenie
BROWSABLE	Dane „przeglądarkowe” (link, e-mail, ...).
DEFAULT	Jeżeli komponent jest domyślnym elementem.
LAUNCHER	Główna aktywność aplikacji.
HOME	Główna aktywność telefonu (przycisk HOME)

46

## Intencje

**EXTRAS** – dodatkowe dane, przechowywane w postaci kolekcji klucz-wartość.  
`putExtra()`

Możliwe jest wykorzystanie obiektu `Bundle` umożliwiającego dodanie całej kolekcji typu klucz – wartość

```
Bundle x = new Bundle();
x.putString("klucz", wartość);
...
intent.putExtras
```

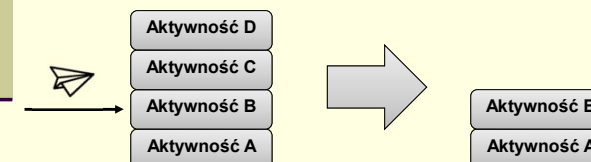


47

## Intencje

**FLAGS** – instruują system jak uruchomić aktywność i jak ją traktować.  
`setFlags(int)`, `addFlags(int)`

`FLAG_ACTIVITY_CLEAR_TOP`

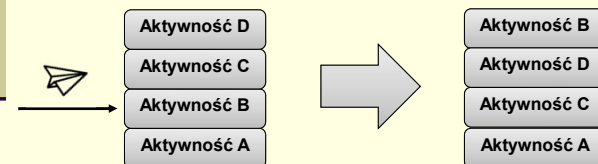


48

## Intencje

FLAGS – instruują system jak uruchomić aktywność i jak ją traktować.

FLAG\_ACTIVITY\_REORDER\_TO\_FRONT



49

## akcje / URI

akcja: dane:

- ACTION\_VIEW content://contacts/id

????

- ACTION\_DIAL content://contacts/id

????

- ACTION\_VIEW tel:123

????

- ACTION\_VIEW <http://www.pwsz.nysa.pl>

????

- ACTION\_VIEW content://contacts/

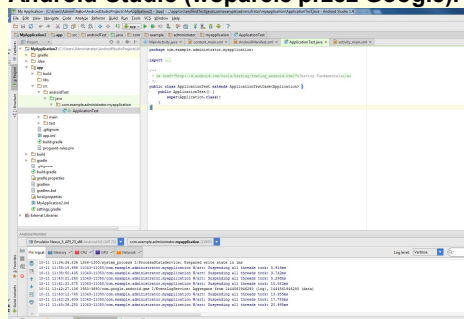
????

50

## Środowiska programistyczne

Dwa najważniejsze środowiska programistyczne:

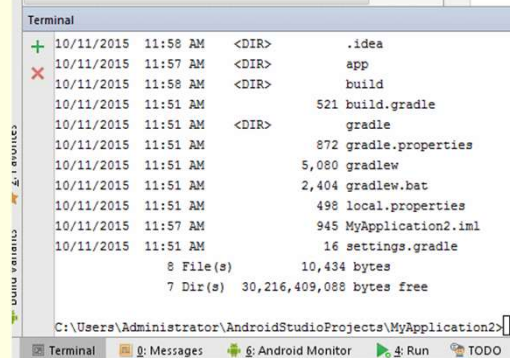
- Eclipse + ADT (stare rozwiązanie),
- Android Studio (wsparcie przez Google).



51

## Środowisko programistyczne

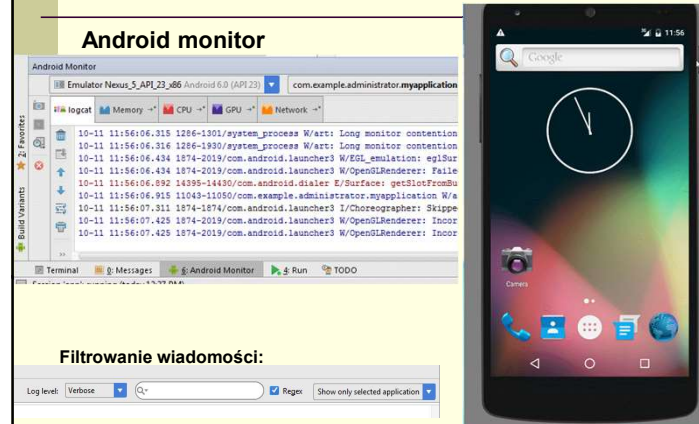
Android Studio – terminal (polecenia windows)



52

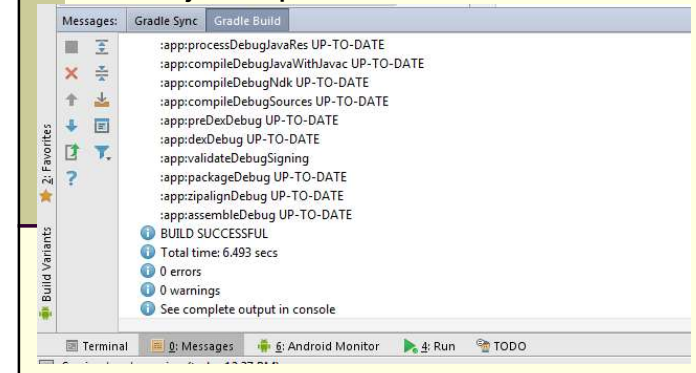
## Środowisko programistyczne

### Android monitor



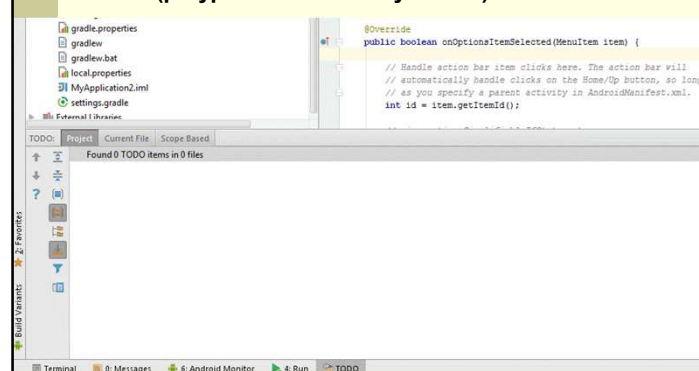
## Środowisko programistyczne

### Informacje z kompilatora:



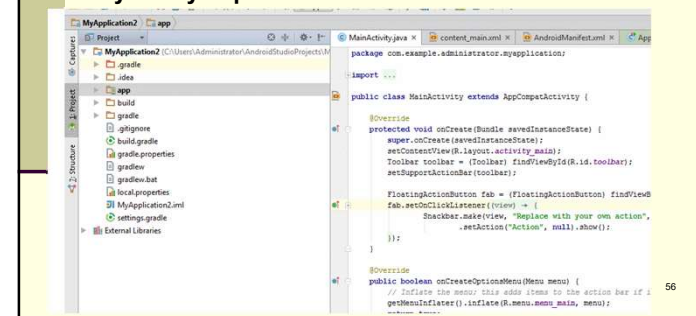
## Środowisko programistyczne

### TODO (przypomina co należy zrobić):

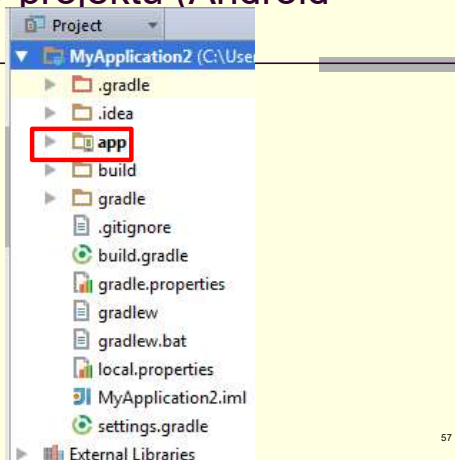


## Foldery projektu (Android studio)

środowisko Android studio przedstawia projekt w postaci różnych widoków – „minimalny zbiór” użytecznych plików - Android



## Foldery projektu (Android studio)

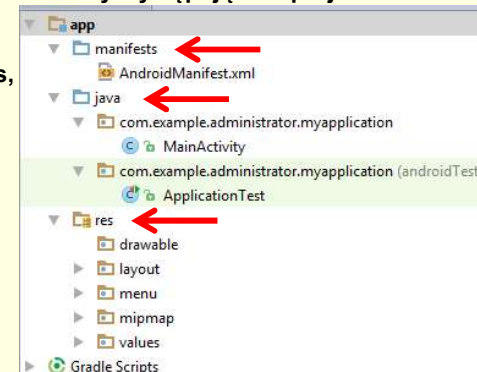


57

## Foldery projektu (Android studio)

Najważniejsze foldery występujące w projekcie aplikacji

Android:  
- manifests,  
- java,  
- res

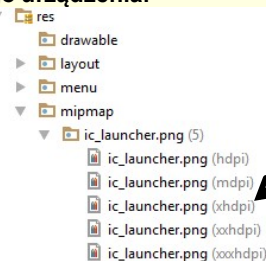


## Foldery projektu (Android studio)

RES – zasoby – są to stałe elementy projektu Android.

Wygoda ze względu na przystosowanie aplikacji do konkretnego urządzenia.

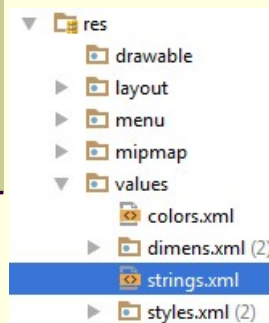
Przykład



w zależności od dpi  
ekranu urządzenia!

## Foldery projektu (Android studio)

RES/VALUES – wartości – elementy z których może korzystać nasza aplikacja:



## Logowanie komunikatów

`Log.v()`, `Log.d()`, `Log.i()`, `Log.w()`, `Log.e()` – funkcje generują wyjście komunikaty w okienku logcat (ERROR, WARN, INFO, DEBUG, VERBOSE).

Składania:

`Log.d("TAG", "KOMUNIKAT");`

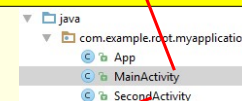
Metoda aplikacji:

```
public void onCreate() {
    Log.d("MyTag", "kom");
    toast.show();
    super.onCreate();
}
```



## Przykład – intencja typu EXPLICIT

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("Wyklad", "Aktywność 1");
    Intent i = new Intent();
    i.setClass(this, com.example.root.myapplication.SecondActivity.class);
    startActivity(i);
}
```



```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("Wyklad", "Aktywnosc 2");
}
```

62

## Przykład – intencja typu EXPLICIT

Plik manifestu:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
  <application
    ...>
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category
          android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
      <activity android:name=".SecondActivity" />
    </application>
  </manifest>
```

## Przykład – intencja typu EXPLICIT

Plik manifestu:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
  <application
    ...>
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category
          android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
      <activity android:name=".SecondActivity" />
    </application>
  </manifest>
```



## Przykład – intencja typu EXPLICIT



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("Wyklad", "Aktywność 1");
    Intent i = new Intent();

    i.setClass(this, com.example.root.myapplication.SecondActivity.class);
    //ALTERNATYWNIIE:
    i.setComponent(new ComponentName(
        "com.example.root.myapplication",
        "com.example.root.myapplication.SecondActivity"));
    i.setClassName(this,
        "com.example.root.myapplication.SecondActivity");
    startActivity(i);
}
```

65

## Intencje typu implicit



Plik manifestu:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category
                android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
        <activity android:name=".SecondActivity">
            <intent-filter>
                <action android:name="moja.akcja"/>
                <category
                    android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

## Przykład – intencja typu IMPLICIT



```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("Wyklad", "Aktywność 1");
    Intent i = new Intent();
    i.setAction("moja.akcja");
    startActivity(i);
}
```

D/Wyklad: Aktywność 1

D/Wyklad: Aktywnosc 2 - IMPLICIT

pr

```
Log.d("Wyklad", "Aktywność 2 - IMPLICIT");
```

67

## Aktywność



**Aktywność (activity):**

Związana z oknem aplikacji (każde okno jest oddzielną aktywnością).

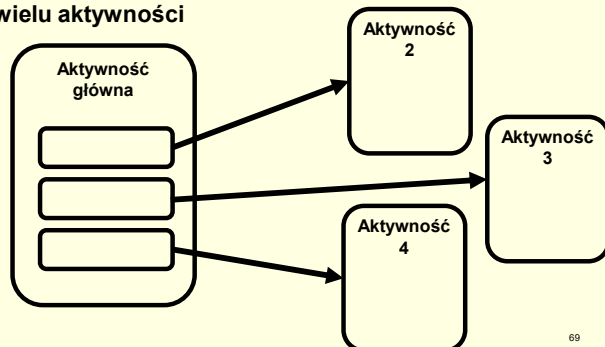
**W danym momencie użytkownik może być w interakcji wyłącznie z jedną aktywnością.**

Android śledzi uruchomione obiekty **Activity** i szereguje je (stos aktywności). Gdy zostaje uruchomiona nowa aktywność, aktywność na wierzchołku stosu jest wstrzymywana. Po zakończeniu działania aktywności ze stosu, wznawiana jest kolejna.

68

## Aktywność

Aplikacja wielookienkowa w Android składa się z wielu aktywności



69

## Aktywność

Możliwe jest określenie zezwoleń koniecznych w celu uruchomienia aktywności.

**Plik manifestu (Aktywność wywoływana):**

```

<activity android:name=".SecondActivity"
  android:permission="com.example.root.myapplication.MOJE_ZEZWOLENIE">
  ...
</activity>
  
```

**Plik manifestu (Aplikacja wywołująca):**

```

<manifest>
  <uses-permission
    android:name="com.example.root.myapplication.MOJE_ZEZWOLENIE"/>
</manifest>
  
```

70

## Cykl życia aktywności w Android

Klasa Activity udostępnia metody cyklu życia wywoływane w momentach:

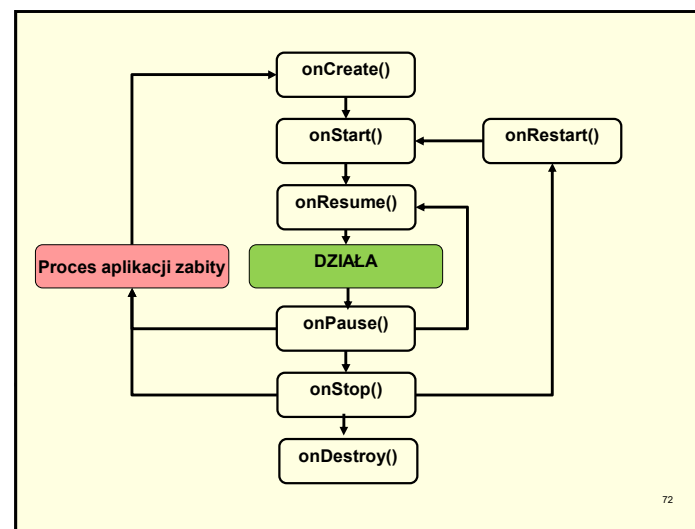
- Uruchomienia,
- Wstrzymania,
- Ponownego uruchomienia aplikacji,
- ...



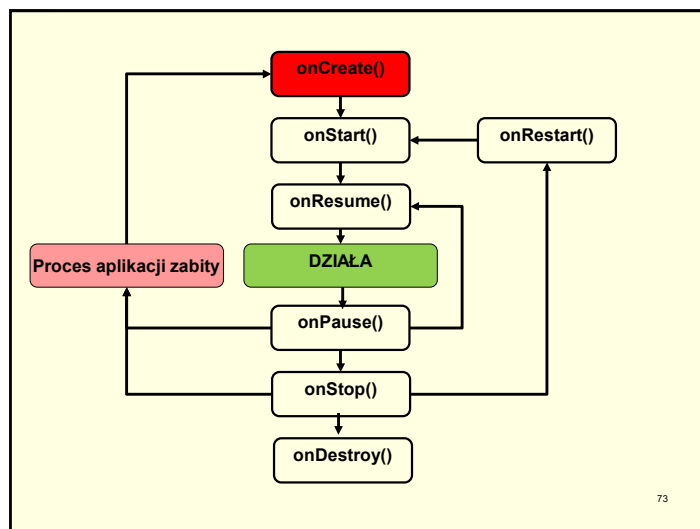
Aplikacje mogą znajdować się w jednym z trzech stanów:

- Aktywnym (pierwszoplanowa),
- Wstrzymanym (aplikacja zasłonięta, nie przyjmuje danych),
- Zatrzymanym (gdy aplikacja całkowicie ukryta).

71



72



## onCreate

Wywoływana jako pierwsza metoda po uruchomieniu aktywności.

W metodzie `onCreate` wykonuje się operacje typowe dla konstruktora (skonfigurowanie głównego okna za pomocą `setContentView`, dodanie dla przycisków metod obsługi zdarzeń, ...).

Metoda przyjmuje parametr `savedInstanceState`, klasy `Bundle`, zawierający zachowany stan aplikacji.

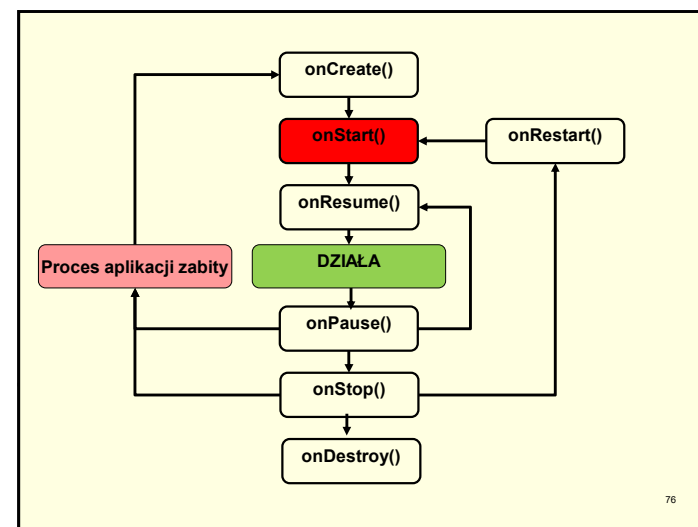


74

```

String stanAktywnosci;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); //Obiekt nadrzędny
    //Odtworzenie stanu aktywności:
    if (savedInstanceState != null) {
        stanAktywnosci = savedInstanceState.getString("KLUCZ");
    }
    //Załadowanie układu: res/layout/main_activity.xml
    setContentView(R.layout.main_activity);
    ...
}
//Metoda wywoływana tylko, gdy istnieje zapisany stan aktywności
//parametr - ten sam obiekt co w metodzie onCreate
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    stanAktywnosci = savedInstanceState.getString("KLUCZ");
}
//Metoda wywoływana przy tymczasowym usuwaniu aktywności
@Override
public void onSaveInstanceState(Bundle state) {
    state.putString("KLUCZ", stanAktywnosci);
    super.onSaveInstanceState(state);
}
}

```



## onStart

Wywoływana w chwili, gdy aktywność **staje się widoczna** dla użytkownika – proces ten może się powtarzać wielokrotnie (użytkownik może przełączać aktywności).

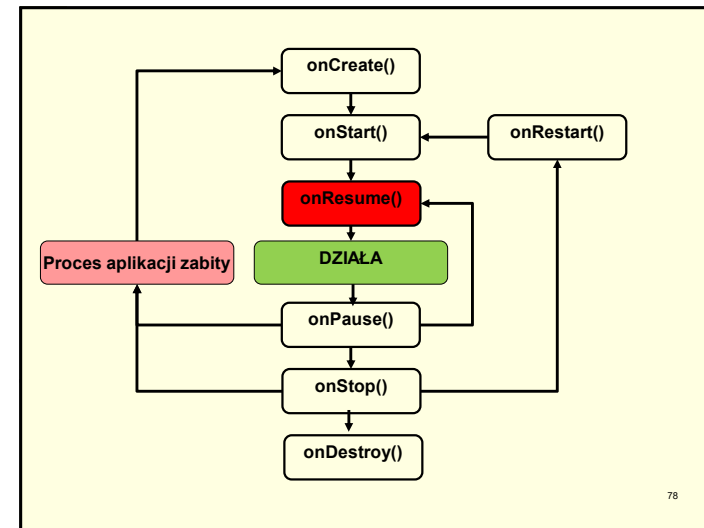


Może zawierać inicjalizację kodu, który obsługuje interfejs graficzny,



rejestracja Broadcast Listener'a, który reaguje na komunikaty zmieniając stan UI,  
...

77



78

## onResume

Wywoływana w chwili, gdy aplikacja **staje się interaktywna** dla użytkownika (przesuwa się na pierwszy plan).



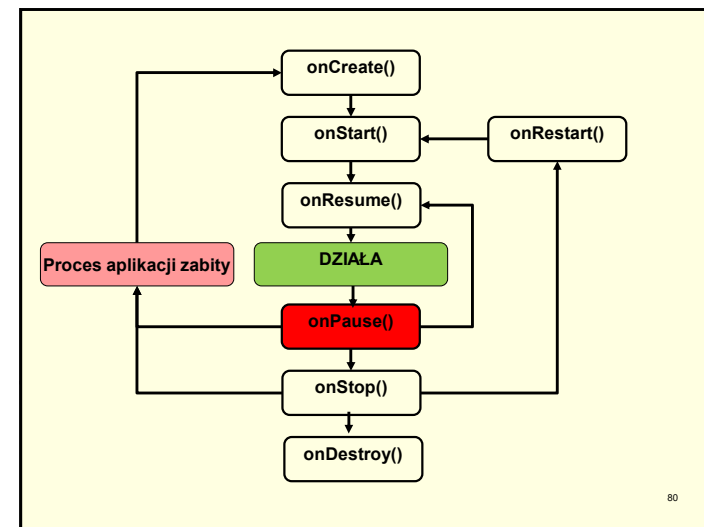
metoda ta jest odpowiednim elementem np. do rozpoczęcia odtwarzania multimedialnych.



```

@Override
public void onResume() {
    super.onResume(); //W pierwszej kolejności należy wywołać
                      //metodę obiektu nadrzędnego.
    //np. gdy obiekt animacja jest zwalniany przy utracie aktywności:
    if (animacja == null) {
        initializeAnimation();
    }
}

```



80

## onPause

Uruchamiana przy **pierwszej** oznace, że **użytkownik opuszcza (traci Focus)** aktywność (nie zawsze oznacza to zakończenie aktywności).

Odtwarzanie multimediiów, animacji powinno zostać zapauzowane (z możliwością szybkiego ich wznowienia).

**Możliwe przyczyny:**

- Zdarzenia przerywające aplikację (telefon, wyłączenie ekranu, nawigacja do innej aktywności),
- Przy odpalonych wielu aplikacjach – tylko aktywna posiada ekran (reszta zapauzowane),
- Wywołanie okna dialogowego – część aktywności widoczna, ale traci „focus”.



## onPause

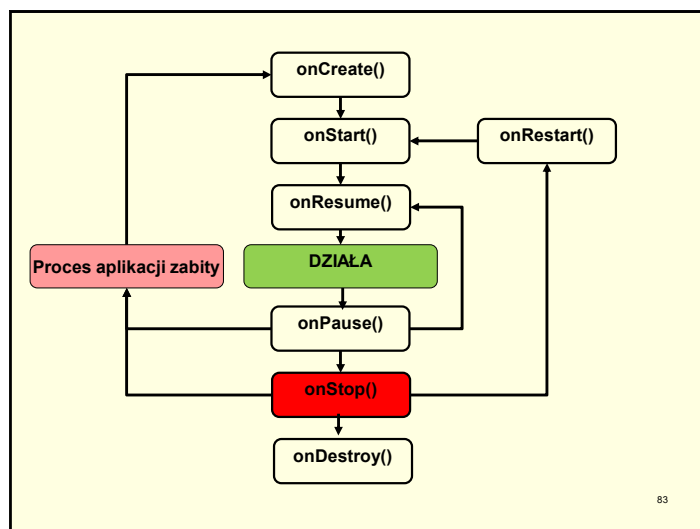
W metodzie obsługi powinno nastąpić zwolnienie zasobów wyczerpujących baterię (GPS, broadcast receivers, ...).

Metoda powinna wykonywać **TYLKO KRÓTKIE CZYNNOŚCI** (np. nie zapisywać stanu).



```
@Override
public void onPause() {
    super.onPause(); //Zawsze powinno się wywoływać metodę klasy
                    //nadrzędnej pierwszej

    //Zwolnienie zasobu
    if (Kamera != null) {
        Kamera.release();
        Kamera = null;
    }
}
```



83

## onStop

Wywoływana w przypadku, gdy aktywność staje się **całkowicie niewidoczna** lub gdy kończy swoje działanie.

Powinno się zwalniać wszystkie zaalokowane zasoby (np. Broadcast receiver'y), zaalokowaną pamięć.

Metoda powinna wykonywać operacje obciążające urządzenie (w przeciwieństwie do poprzedniej) – np. zapis do bazy danych..

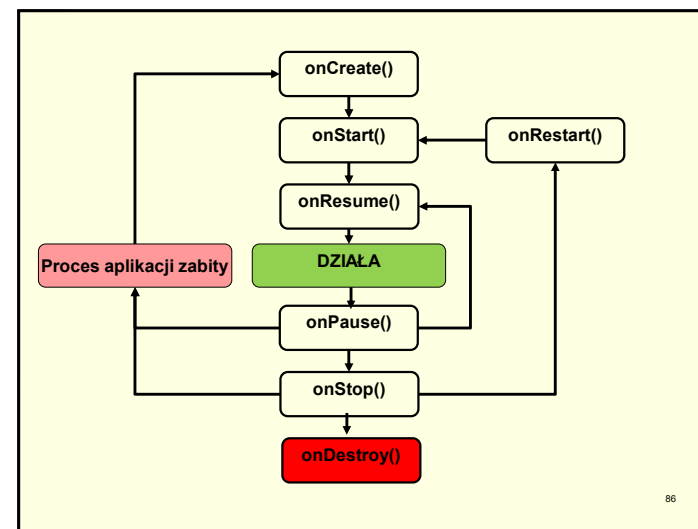


84

## onStop

Gdy aktywność znajduje się w stanie zatrzymanym **system może zabić jej proces** gdy brakuje miejsca w pamięci (system zapamiętuje stan aplikacji w obiekcie Bundle).

```
@Override
protected void onStop() {
    super.onStop();
    //Wykonanie operacji obciążających urządzenie
    saveToDatabase();
    //Zwolnienie innych zasobów:
    broadcastReceiversStop();
}
```



86

## onDestroy

Wywoływana **przed zabiciem** aktywności.

Przyczyny:

- Kończenie pracy aktywności,
- Z powodu braku pamięci,
- Zmiana orientacji ekranu (po niej niezwłocznie zostanie wywołana metoda onCreate).

Powinno się tu zwolnić wszystkie zasoby, które nie zostały zwolnione we wcześniejszych metodach.



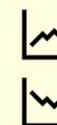
87

## Usuwanie z pamięci

System nigdy nie zabija samej aktywności – zabija cały proces, w którym aktywność się wykonuje (np. przy niedoborze pamięci).

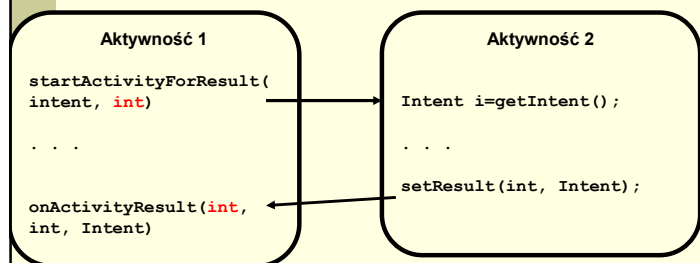
Prawdopodobieństwo zabicia procesu

Prawdopodobieństwo zabicia procesu	Stan procesu	Stan aktywności
NAJMNIEJSZE	PIERWSZOPLANOWY	CREATED, STARTED, RESUMED
ŚREDNIE	TŁA (NIEWIDOCZNY)	PAUSED
NAJWIĘKSZE	TŁA (NIEWIDOCZNY)	STOPPED, DESTROYED



## Wynik działania aktywności

Jeżeli aktywność ma zwracać daną, należy wykorzystać schemat działania (**int** – parametr pozwalający rozróżnić aktywności wyniki powracające z aktywności potomnych):



### AKTYWNOŚĆ GŁÓWNA:

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("Wyklad", "Aktywność 1");
    Intent i = new Intent();
    i.putExtra("tlumacz", "czerwony");
    i.setClass(this,
com.example.root.myapplication.SecondActivity.class);
    startActivityForResult(i, 1);
    i.putExtra("tlumacz", "POM");
    startActivityForResult(i, 1);
}

protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
    if(requestCode==1){
        if(resultCode==RESULT_OK){
            Log.d("Wyklad", data.getStringExtra("slowo"));
        } else{
            Log.d("Wyklad", "NIEZNANE SLOWO");
        }
    }
}
  
```

## Wynik działania aktywności

### AKTYWNOŚĆ 2:

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("Wyklad", "Aktywnosc 2 - TLUMACZ");
    Intent i=getIntent();
    Intent odp= new Intent();
    if(i.getStringExtra("tlumacz").equals("czerwony")){
        odp.putExtra("slowo", "red");
        setResult(Activity.RESULT_OK,odp);
        finish();
    } else{
        setResult(Activity.RESULT_CANCELED);
        finish();
    }
}
  
```

## Wynik działania aktywności

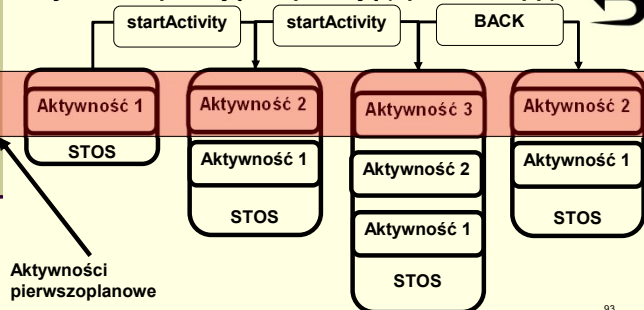
### AKTYWNOŚĆ 2:

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("Wyklad", "Aktywnosc 2 - TLUMACZ");
    Intent i=getIntent();
    Intent odp= new Intent();
    if(i.getStringExtra("tlumacz").equals("czerwony")){
        odp.putExtra("slowo", "red");
        setResult(Activity.RESULT_OK,odp);
        finish();
    } else{
        setResult(Activity.RESULT_CANCELED);
        finish();
    }
}
  
```

## Zadania i stos aktywności

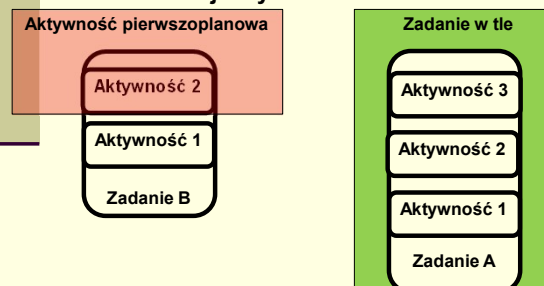
Zadanie jest kolekcją aktywności, którą wywołuje użytkownik pracując z aplikacją (np. e-mail app).



93

## Zadania i stos aktywności

Po przyciśnięciu przycisku HOME użytkownik może uruchomić inną aplikację. Poprzednie zadanie wędruje do tła, powstaje nowy stos dla uruchomionej aktywności.



94

## Zadania i stos aktywności

Wykorzystując plik Manifestu lub flagi przy uruchamianiu aktywności można zmodyfikować domyślny system zarządzania zadaniami.

Flaga	Znaczenie
FLAG_ACTIVITY_NEW_TASK	Uruchamia aktywność w nowym zadaniu
FLAG_ACTIVITY_SINGLE_TOP	Jeżeli uruchamianą aktywnością jest aktywność bieżąca (na szczycie), to do stosu nie jest dodawany nowy element (Zamiast stanu A-B-C-C uzyskujemy A-B-C)
FLAG_ACTIVITY_CLEAR_TOP	Usuwa ze stosu aktywności położone wyżej

95

## Fragmenty



Fragmenty stanowią część interfejsu użytkownika w aktywności.

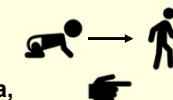
Można połączyć kilka fragmentów w celu stworzenia UI w jednej aktywności.



Pojedynczego fragmentu można również używać w wielu aktywnościach.



Fragmenty posiadają swój własny cykl życia, obsługują własne zdarzenia, można je dodawać i usuwać w działającej aktywności.

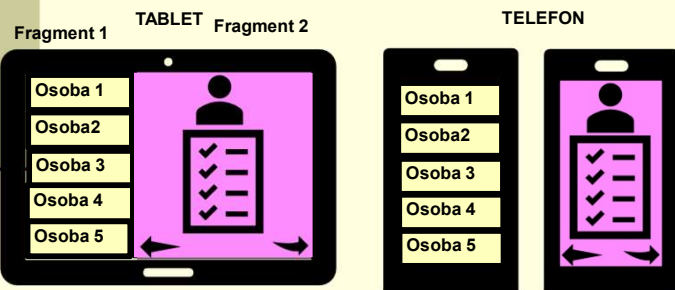


96



## Fragmenty

Fragmenty umożliwiają wygodniejsze projektowanie UI dla np. różnej wielkości ekranów



## Cykl życia fragmentu

Cykl życia aktywności, w której uruchomiony jest fragment ma bezpośredni wpływ na cykl życia fragmentu.

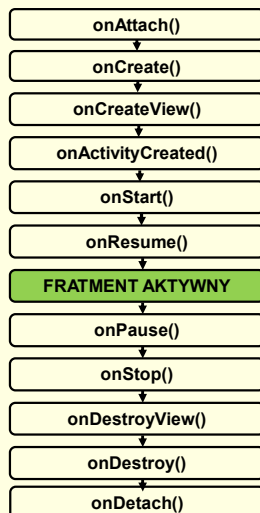
Gdy w aktywności wywołana jest metoda `onPause()`, dla każdego fragmentu aktywności również wywołana jest ta metoda.

Występują metody o tej samej nazwie aby łatwo skonwertować aktywność na fragment:

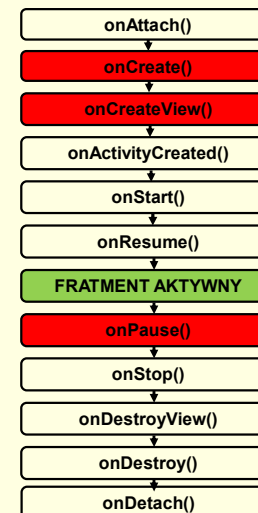
`onCreate()`, `onStart()`, `onPause()`, `onStop()`



98



99



100

## Cykl życia fragmentu

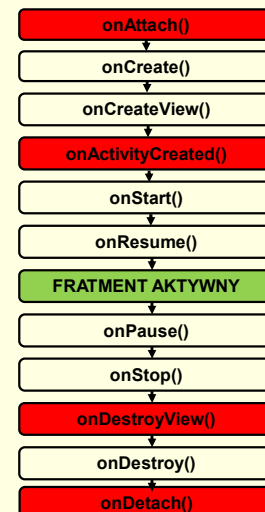
**onCreate** – metoda wywoływana przy tworzeniu fragmentu. Powinna następować tu inicjalizacja kluczowych komponentów, które się nie zmieniają w przypadku **onPause** lub **onStopped**.

**onCreateView** – wywoływana w przypadku, gdy fragment „rysuje” swój interfejs graficzny po raz pierwszy. W przypadku, gdy fragment posiada UI metoda musi zwrócić obiekt klasy **View**, null – gdy fragment nie dostarcza UI.

**onPause** – wywoływana, gdy użytkownik opuszcza fragment.



101



102

## Cykl życia fragmentu

**onAttach** – wywoływana, gdy fragment został skojarzony z aktywnością

**onActivityCreated** – zaraz po wyjściu z metody **onCreate** aktywności

**onDestroyView** – gdy hierarchia widoków związana z fragmentem jest usuwana

**onDetach** – gdy fragment jest usuwany z aktywności



103

## Fragmenty

Aby stworzyć fragment należy dziedziczyć po klasie **Fragment** lub podklasach:

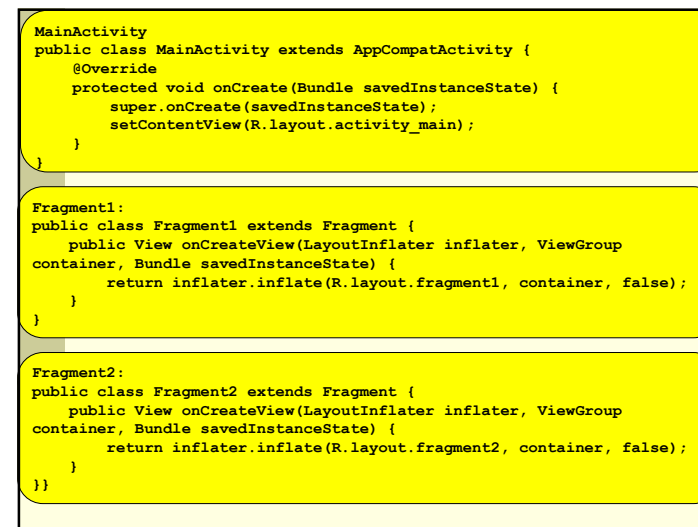
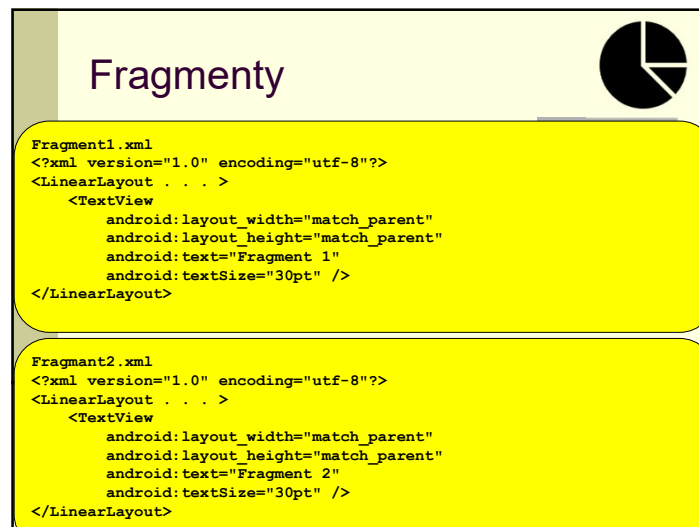
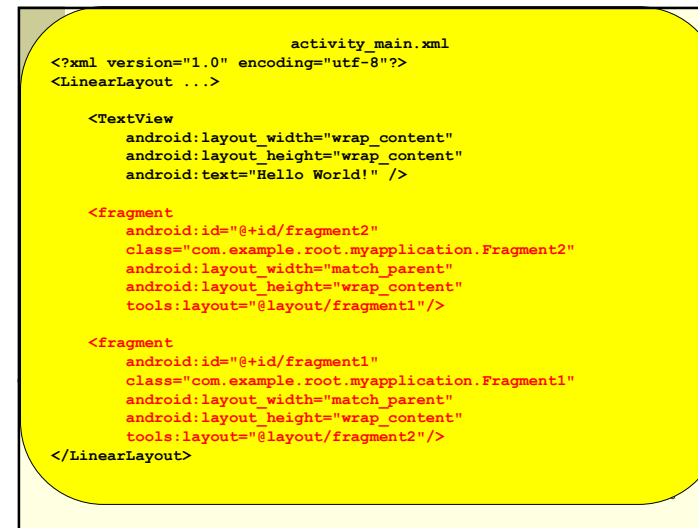
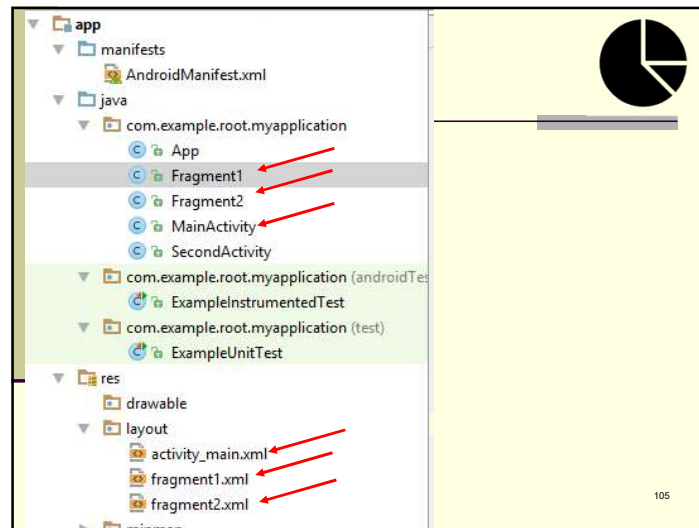
**DialogFragment** – odpowiednik okna dialogowego.

**ListFragment** – odpowiednik listy wyboru.

**PreferenceFragment** – odpowiednik okienka z ustawieniami.



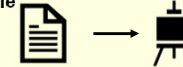
104





## Fragmenty

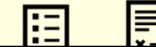
LayoutInflater – obiekt umożliwiający utworzenie obiektu klasy View z pliku XML z layoutem (z wykorzystaniem metody inflate).



```
LayoutInflater inflater=getLayoutInflater();
lub
LayoutInflater inflater = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
```

```
View v = inflater.inflate(int resource, ViewGroup
parent, boolean attachToRoot)
```

resource – ID Layout'u (z pliku zasobów),  
parent – Layout, do którego dodany zostanie resource,  
attachToRoot – czy dodawany element będzie elementem potomnym znacznika głównego parent, czy nie



## Fragmenty

### Dodawanie fragmentu dynamicznie

W celu dodania, usunięcia, zamiany fragmentów działającej aplikacji należy wykorzystać obiekt **FragmentManager**

```
public class MainActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        FragmentManager manager = getSupportFragmentManager();
        FragmentTransaction transaction = manager.beginTransaction();
        //OPERACJE DODAWANIA/USUWANIA/PODMIANY FRAGMENTÓW
        transaction.commit();
    }
}
```

```
activity_main.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:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/fragment1"
        android:orientation="vertical">
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/fragment2"
        android:orientation="vertical">
    </LinearLayout>
</LinearLayout>
```

## Fragmenty

```
Fragment1.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout . . . >
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Fragment 1"
        android:textSize="30pt" />
</LinearLayout>
```

```
Fragment2.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout . . . >
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Fragment 2"
        android:textSize="30pt" />
</LinearLayout>
```

## Fragmenty

```
Fragment1:
public class Fragment1 extends Fragment {
    public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment1, container, false);
    }
}
```

```
Fragment2:
public class Fragment2 extends Fragment {
    public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment2, container, false);
    }
}
```

114

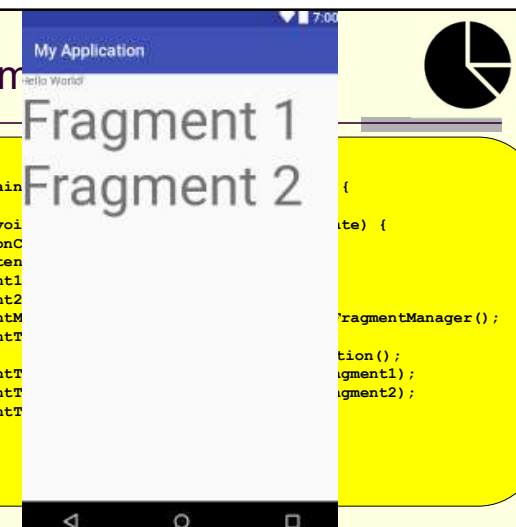
## Fragmenty

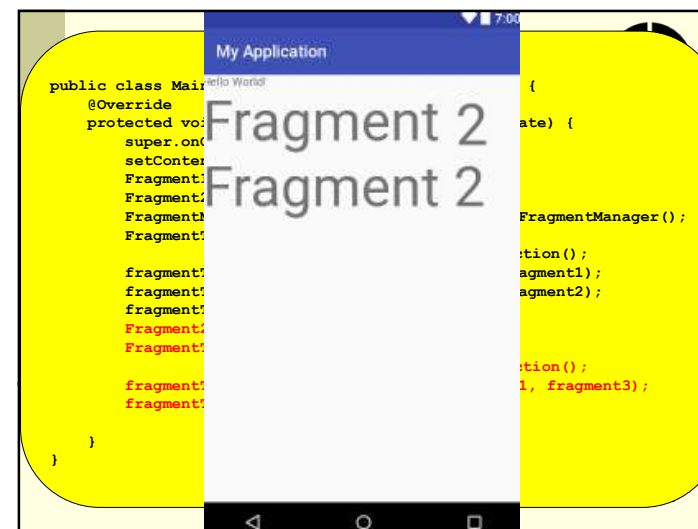
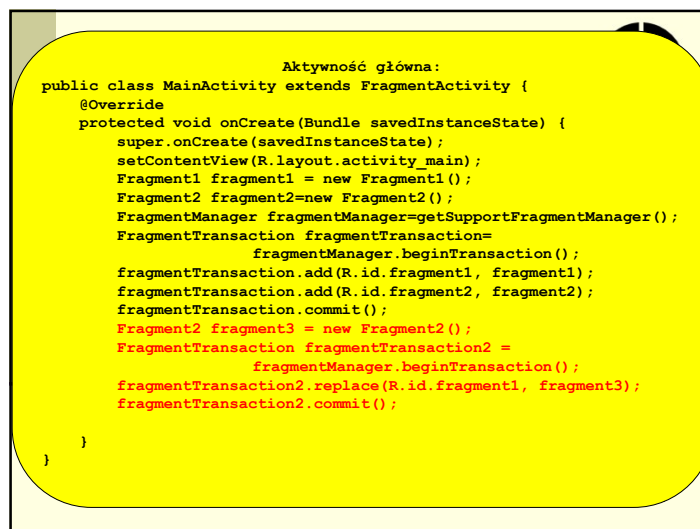
**Aktywność główna:**

```
public class MainActivity extends FragmentActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Fragment1 fragment1 = new Fragment1();
        Fragment2 fragment2 = new Fragment2();
        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction fragmentTransaction =
            fragmentManager.beginTransaction();
        fragmentTransaction.add(R.id.fragment1, fragment1);
        fragmentTransaction.add(R.id.fragment2, fragment2);
        fragmentTransaction.commit();
    }
}
```

## Fragmenty

```
public class MainActivity extends FragmentActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Fragment1 fragment1 = new Fragment1();
        Fragment2 fragment2 = new Fragment2();
        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction fragmentTransaction =
            fragmentManager.beginTransaction();
        fragmentTransaction.add(R.id.fragment1, fragment1);
        fragmentTransaction.add(R.id.fragment2, fragment2);
        fragmentTransaction.commit();
    }
}
```





## Zasoby

Wydzielenie wyświetlanych elementów aktywności:

- tekstów,
- obrazków,

do zasobów umożliwia ich internalizację (różna zawartość w zależności od lokalizacji) oraz zapewnienie innych elementów w zależności od wielkości/rozdzielczości ekranu.

Zasoby znajdują się w folderze /res aplikacji.

Zasoby domyślne – niezależne od konfiguracji/lokalizacji,  
Zasoby alternatywne – zależne od konfiguracji/lokalizacji (odpowiedni kwalifikator w nazwie folderu).

## Zasoby

res/layout/

res/layout-land/

## Zasoby

### Wybrane podfoldery /res (ZASOBY DOMYŚLNE):

Folder	Opis
color	Kolory
drawable	.png, .9.png, .jpg, .gif, kształty, ...
mipmap	Ikony
layout	Pliki układów
menu	Różne menu aplikacji
values	Wartości proste (napisy, liczby, kolory, ...)
xml	Pliki XML, które będą odczytywane w trakcie działania aplikacji
font	Pliki czcionek .ttf, .otf, .ttc

Ciekawostka – umieszczenie pliku z zasobem bezpośrednio w folderze res spowoduje błąd kompilacji.

121

## Zasoby

W trakcie działania programu system wykrywa bieżącą konfigurację i ładuje w pierwszej kolejności zasoby alternatywne (zgodne z konfiguracją).



W celu stworzenia zasobów alternatywnych w folderze res należy umieścić podfolder:

`<nazwa_domyślna>-<kwalifikator_konfiguracji>.`

Kwalifikator konfiguracji może wystąpić wielokrotnie (kolejne kwalifikatory oddzielone kreską).

np.  
drawable-port-hdpi

122

## Zasoby

### Wybrane kwalifikatory:

Kwalifikator	Przykład	Opis
Język i region	en, pl, en-rUS, fr-rCA	Język – dwie litery zgodne z ISO 939-1, region „r” i dwie litery z ISO 3166-1-alpha-2
Kierunek układu	ldrtl, ldltr	Z prawej do lewej, z lewej do prawej (domyślny)
Rozmiar ekranu	small, normal, large, xlarge	
Orientacja ekranu	port, land	Wertykalna, horyzontalna
Screen pixel density (dpi)	ldpi, mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi, nodpi, tvdpi, anydpi	ldpi: ok. 120dpi, mdpi: ok. 160dpi, hdpi: ok. 240dpi, xhdpi: 320dpi, xxhdpi: ok. 480dpi, xxxhdpi: ok. 640dpi
Tryb nocny	night, notnight	

Ważna jest kolejność umieszczania kwalifikatorów (np. dla opcji powyżej – w kolejności ich umieszczenia – pełna informacja w dokumentacji Google).

## Zasoby



### Logiczne – res/values/bools.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <bool name="a">true</bool>
  <bool name="b">true</bool>
</resources>
```

### Kolory – res/values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="czerwony">#f00</color>
  <color name="prz_czerwony">#80ff0000</color>
</resources>
```

#RGB - #00F – kolor niebieski (12bit)  
 #ARGB - #800F – przezroczystość 50% (12bit)  
 #RRGGBB - #FF00FF – kolor purpurowy (24bit)  
 #AARRGGBB - #80FF00FF – kolor purpurowy, 50% przezroczystości (24bit)

124

## Zasoby



### Rozmiary – res/values/dimens.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <dimen name="wysokosc">25px</dimen>
  <dimen name="dlugosc">30dp</dimen>
  <dimen name="czcionka">18sp</dimen>
</resources>
```

Jednostka	opis
px, in, mm, pt	piksele, cale, milimetry, punkty (1/72 cala)
dp	Jednostka bazuje na wyswietlaczu 160dpi (160 punktów na cal). 1dp jest odpowiednikiem 1px przy gęstości 160dpi. Stosunek ten jest zmieniany wraz ze zmianą gęstości wyswietlacza.
sp	Jednostka jest zależna od wartości Settings.System.FONT_SCALE (powinna być stosowana do ustalania wielkości czcionek).

## Zasoby



### Liczby całkowite i tablice liczb całkowitych – res/values/integers.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <integer name="a">75</integer>
  <integer name="b">5</integer>
  <integer-array name="wartosci">
    <item>1</item>
    <item>2</item>
    <item>3</item>
    <item>4</item>
  </integer-array>
</resources>
```

### Napisy – res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="a">napis 1</integer>
  <string name="b">napis 2</integer>
</resources>
```

126

## Zasoby



Do niektórych zasobów możliwe jest tworzenie aliasów (mechanizm nie wspierają animation, menu, raw, xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="a">napis 1</integer>
  <string name="b">%string/a</integer>
</resources>
```

W przypadku układów:

```
<?xml version="1.0" encoding="utf-8"?>
<merge>
  <include layout="@layout/main"/>
</merge>
```

127

## Zasoby



Dostęp do zasobów:

- W kodzie poprzez klasę R (automatycznie generowany kod w wyniku kompilacji) – np. R.string.a

Wyrażenie: **R.<typ zasobu>.<nazwa zasobu>**

```
setContentView(R.layout.main);
TextView x = (TextView) findViewById(R.id.msg);
x.setText(R.string.a);
```

- W XML - @string/a

Wyrażenie: **@<typ zasobu>.<nazwa zasobu>**

```
<?xml version="1.0" encoding="utf-8"?>
<EditText ...
  android:textColor="@color/red"
  android:text="@string/a" />
```

128



## Kontekst aplikacji



Kontekst aplikacji – obiekt `Context` jest elementem zapewniającym dostęp do głównych funkcjonalności. Klasa `Context` wykorzystywana jest do zarządzania informacjami związanymi z konfiguracją oraz danymi aplikacji (zasoby, preferencje ...).

`Context c = getApplicationContext();` - pobiera obiekt `Context` dla aktualnego procesu

`String x = getResources().getString(R.string.x);` - pobranie zapisanego w zasobach aplikacji łańcucha znaków (identyfikator wygenerowany w pliku `R.java`)

129

## Kontekst aplikacji



Uwaga w Android występuje wiele „typów” kontekstu – np.

- Instancja aplikacji jako kontekst
- Activity
  - Instancja aktywności (this)
  - `getApplicationContext()` w Activity (App context)
  - `getBaseContext()` (kontekst przekazany w konstruktorze `ContextWrapper`)
- Fragment
  - `getContext()` w Fragmentcie
- View
  - `getContext()` w View
- Broadcast Receiver
  - Context dostarczony w BR
- Service
  - Instancja usługi (this)
  - `getApplicationContext()` w Service
- Context
  - `getApplicationContext()` w instancji Context



130

## Kontekst aplikacji



Co można zrobić z danym kontekstem:

kontekst > wywołanieV	aplikacji	aktywności	usługi	ContentProvidera	BroadcastReceiwera
Okno dialogowe		X			
aktywność		X			
Layout inflation		X			
Usługa	X	X	X	X	X
Bindowanie do usługi	X	X	X	X	
wysyłanie broadcastu	X	X	X	X	X
rejestracja BR	X	X	X	X	
Ładowanie zasobu	X	X	X	X	X

131

## Kontekst aplikacji



Wszystkie typy kontekstów są „krótkotrwałe” z wyjątkiem kontekstu Aplikacji.

Ten typ kontekstu uzyskujemy z klasy aplikacji lub używając metody `getApplicationContext()`.

132

## Zasoby



### res/values/strings

```
<resources>
  <string name="app_name">My Application</string>
  <string name="tekst1">Wykład</string>
</resources>
```

### res/values/integers

```
<resources>
  <integer name="rok">3</integer>
</resources>
```

### Aktywność główna:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Context context= getApplicationContext();
    String napis=
        context.getResources().getString(R.string.tekst1);
    int rok =context.getResources().getInteger(R.integer.rok);
    Log.d("Wykład", napis+" rok "+rok);
}
```

## Zasoby



### res/values/strings

```
<resources>
```

Debug

tion D/Wykład: Wykład rok 3

```
<integer name="rok">3</integer>
</resources>
```

### Aktywność główna:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Context context= getApplicationContext();
    String napis=
        context.getResources().getString(R.string.tekst1);
    int rok =context.getResources().getInteger(R.integer.rok);
    Log.d("Wykład", napis+" rok "+rok);
}
```

## Zasoby



Zasób	Metoda
Strings	String tekst = getResources().getString(R.string.przyklad_tekst);
Arrays	String [] tekst_tab = getResources().getStringArray(R.array.tablica_str);
Booleans	boolean taknie = getResources().getBoolean(R.bool.taknie);
Integers	int liczba = getResources().getInteger(R.integer.liczba_int);
Colors	int kolor = getResources().getColor(R.color.kolor);
dimens	float wymiar=getResources().getDimension(R.dimen.txtSize);
drawables	ColorDrawable prostokat= (ColorDrawable) getResources().getDrawable(R.drawable.ProstCzerw);

```
<resources>
  <string name="app_name">My Application</string>
  <string name="action_settings">Settings</string>
  <string name="napis">Hello World</string>
</resources>
```

res/strings.xml

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

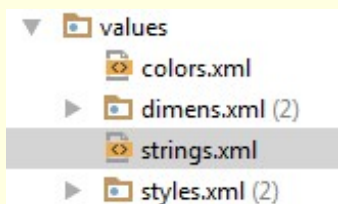
res/layout/content\_main.xml

Hello World

136

## Przykłady zastosowania – aplikacja wielojęzyczna

```
<resources>
  <string name="app_name">My Application</string>
  <string name="action_settings">Settings</string>
  <string name="napis">Hello World</string>
</resources>
```



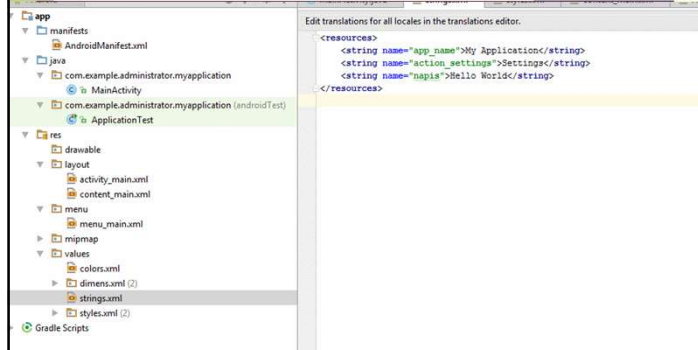
137

## Przykłady zastosowania – aplikacja wielojęzyczna

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

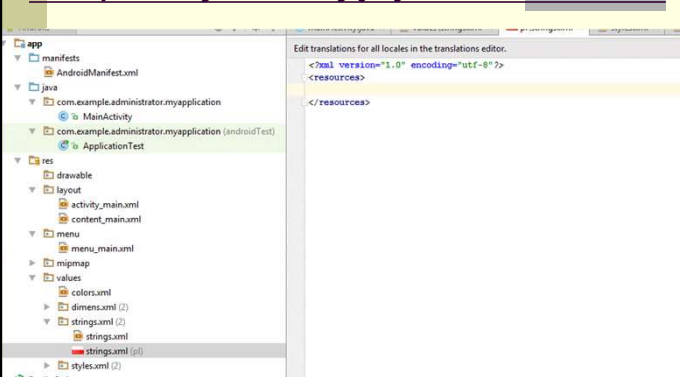
138

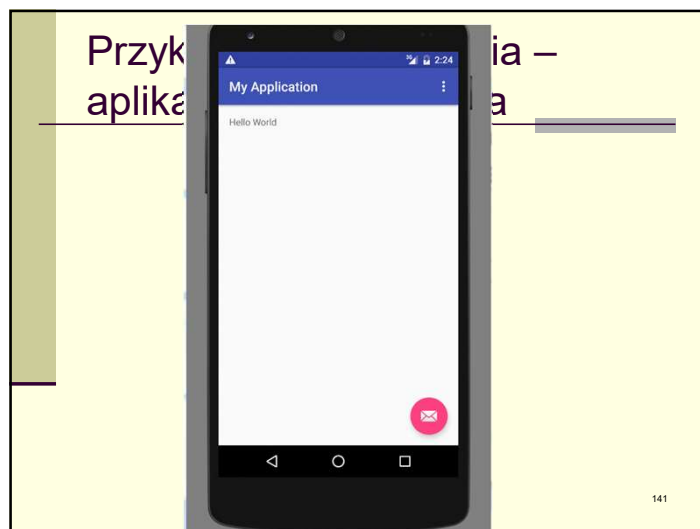
## Przykłady zastosowania – aplikacja wielojęzyczna



139

## Przykłady zastosowania – aplikacja wielojęzyczna





## Zmiana konfiguracji urządzenia

Niektóre ustawienia urządzenia mogą ulec zmianie w trakcie gdy uruchomiony jest program (orientacja ekranu, dostępność klawiatury, język).

Android restartuje aplikację (onDestroy() → onCreate()) – umożliwia to załadowanie alternatywnych zasobów.

Operacja ta może być niekiedy kosztowna (ponowne nawiązanie łączności z serwerem, wykonanie operacji na bitmapie ...)



142

## Zmiana konfiguracji urządzenia

Gdy Android zabija aktywność (w wyniku zmiany konfiguracji), odpowiednio oznaczone fragmenty nie ulegają zniszczeniu (powinno się w nich umieszczać obiekty stanowe np. związane z połączeniem).

Kolejne kroki:

Zadeklarować klasę, która dziedziczy po Fragment (będzie przechowywała referencje do stanowych obiektów).

Wywołać metodę setRetainInstance(true) po utworzeniu fragmentu (oznacza fragment)

Dodać fragment do aktywności

Wykorzystać FragmentManager do pobrania fragmentu po zrestartowaniu aktywności

## Zasoby

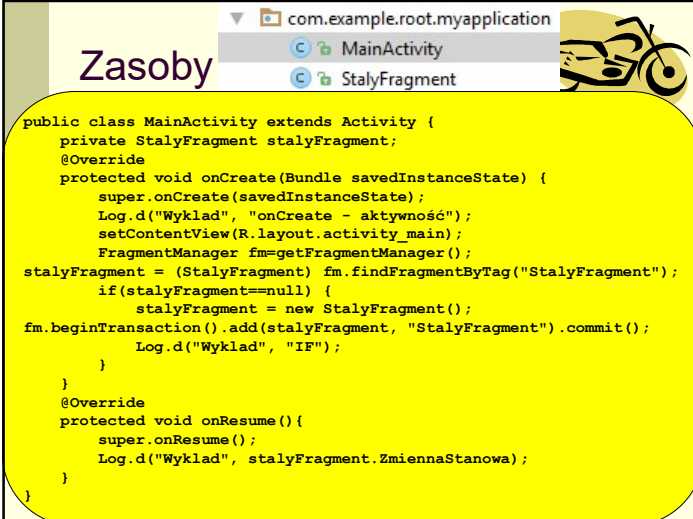


com.example.root.myapplication  
MainActivity  
StalyFragment

```
public class StalyFragment extends Fragment {
    public String ZmiennaStanowa;
    @Override
    public void onCreate(Bundle x) {
        super.onCreate(x);
        ZmiennaStanowa="Skomplikowany stan";
        setRetainInstance(true);
    }
}
```

144

## Zasoby

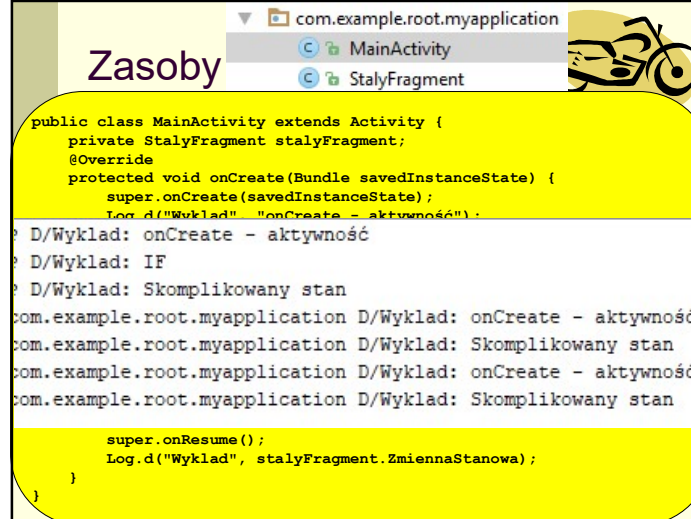


```

public class MainActivity extends Activity {
    private StalyFragment stalyFragment;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("Wyklad", "onCreate - aktywność");
        setContentView(R.layout.activity_main);
        FragmentManager fm=getFragmentManager();
        stalyFragment = (StalyFragment) fm.findFragmentByTag("StalyFragment");
        if(stalyFragment==null) {
            stalyFragment = new StalyFragment();
            fm.beginTransaction().add(stalyFragment, "StalyFragment").commit();
            Log.d("Wyklad", "IF");
        }
    }
    @Override
    protected void onResume() {
        super.onResume();
        Log.d("Wyklad", stalyFragment.ZmiennaStanowa);
    }
}

```

## Zasoby



```

public class MainActivity extends Activity {
    private StalyFragment stalyFragment;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("Wyklad", "onCreate - aktywność");
    }
}

```

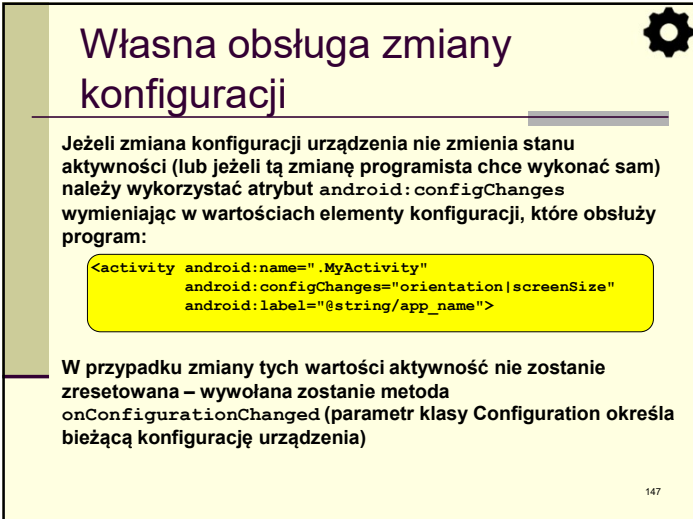
D/Wyklad: onCreate - aktywność  
D/Wyklad: IF  
D/Wyklad: Skomplikowany stan  
com.example.root.myapplication D/Wyklad: onCreate - aktywność  
com.example.root.myapplication D/Wyklad: Skomplikowany stan  
com.example.root.myapplication D/Wyklad: onCreate - aktywność  
com.example.root.myapplication D/Wyklad: Skomplikowany stan

```

        super.onResume();
        Log.d("Wyklad", stalyFragment.ZmiennaStanowa);
    }
}

```

## Własna obsługa zmiany konfiguracji



Jeżeli zmiana konfiguracji urządzenia nie zmienia stanu aktywności (lub jeżeli tą zmianę programista chce wykonać sam) należy wykorzystać atrybut `android:configChanges` wymieniając w wartościach elementy konfiguracji, które obsłuży program:

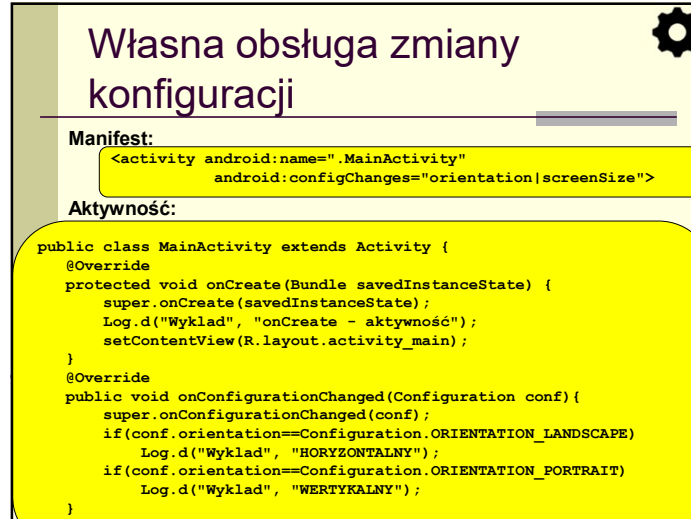
```

<activity android:name=".MyActivity"
    android:configChanges="orientation|screenSize"
    android:label="@string/app_name">

```

W przypadku zmiany tych wartości aktywność nie zostanie zresetowana – wywołana zostanie metoda `onConfigurationChanged` (parametr klasy `Configuration` określa bieżącą konfigurację urządzenia)

## Własna obsługa zmiany konfiguracji



**Manifest:**

```

<activity android:name=".MainActivity"
    android:configChanges="orientation|screenSize">

```

**Aktywność:**

```

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("Wyklad", "onCreate - aktywność");
        setContentView(R.layout.activity_main);
    }
    @Override
    public void onConfigurationChanged(Configuration conf){
        super.onConfigurationChanged(conf);
        if(conf.orientation==Configuration.ORIENTATION_LANDSCAPE)
            Log.d("Wyklad", "HORYZONTALNY");
        if(conf.orientation==Configuration.ORIENTATION_PORTRAIT)
            Log.d("Wyklad", "WERTYKALNY");
    }
}

```

