



PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

WYKŁAD 1

- ACTIVITY
- CYKL ŻYCIA AKTYWNOŚCI

Podstawowe elementy – komponenty, klasy pierwotne z których zbudowana jest aplikacja (VM + Zasoby = APK (Android Package Kit))

Każdy komponent zapewnia konkretną funkcjonalność i posiada wyodrębniony cykl życia. Komponenty działają kooperacyjnie wspólnie zapewniając ukończenie określonego zadania aplikacji

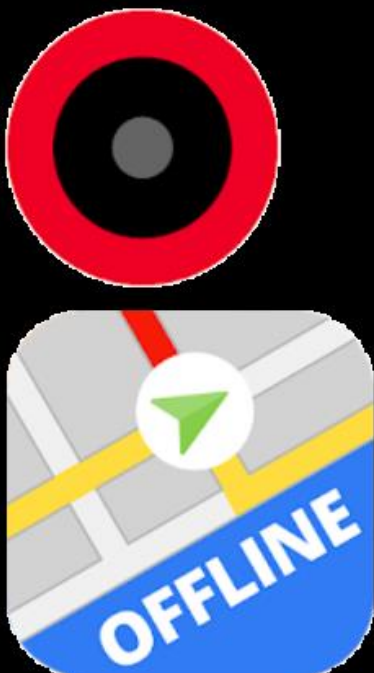
Podstawowe elementy:

1. Activities – pojedynczy ekran z UI
2. Services – long running tasks in the background
3. Broadcast Receiver – responds to system-wide announcements
4. Content Provider

- Activity Class – obiekt, pojedyncze GUI, które poza wyświetlaniem / zbieraniem danych zapewnia pewną funkcjonalność
- Typowo aplikacje zawierają jeden lub więcej obiektów *Activity*
- Aplikacje muszą wyznaczyć jedną czynność jako *main task/entry point*. Ta czynność jest uruchamiana jako pierwsza gdy aplikacja jest włączana
- Czynność może przekazać kontrolę i dane do innej czynności poprzez protokół komunikacji międzyprocesowej *intents* - proces logowania



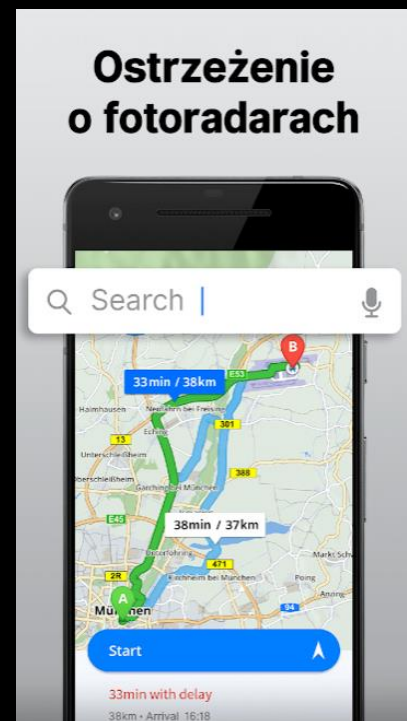
- Service Class – specjalny typ aktywności – nie posiada *Visual User Interface*. Usługa może być aktywna w tle
- Usługi zazwyczaj pracują w tle wykonując „busy-work” przez nieokreślony /nieskończony czas
- Aplikacje rozpoczynają własne usługi lub łączą się z usługami już aktywnymi (GPS, przełączanie między aplikacjami, aktywnościami, brak okienek)



Wirtualna Polska Media S.A.

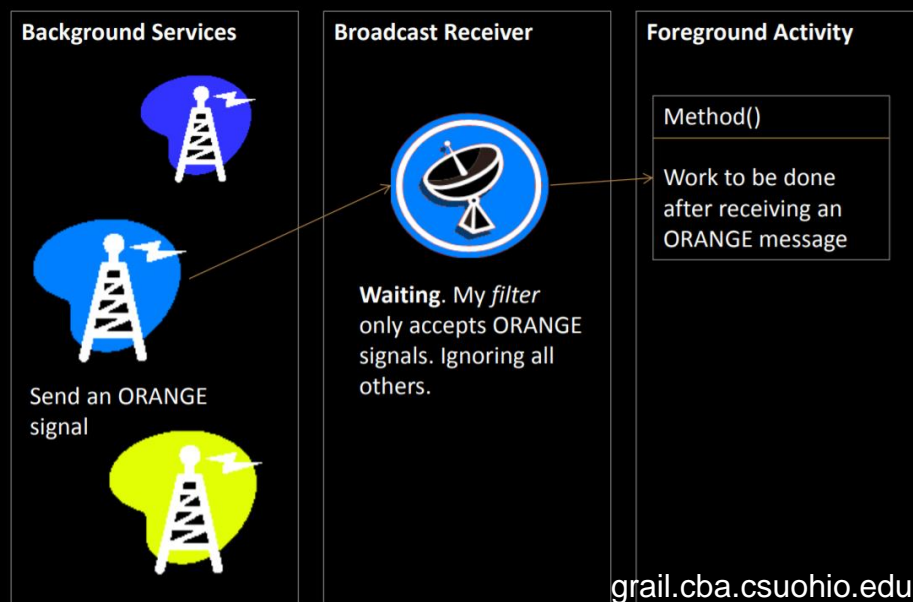


SYBO Games

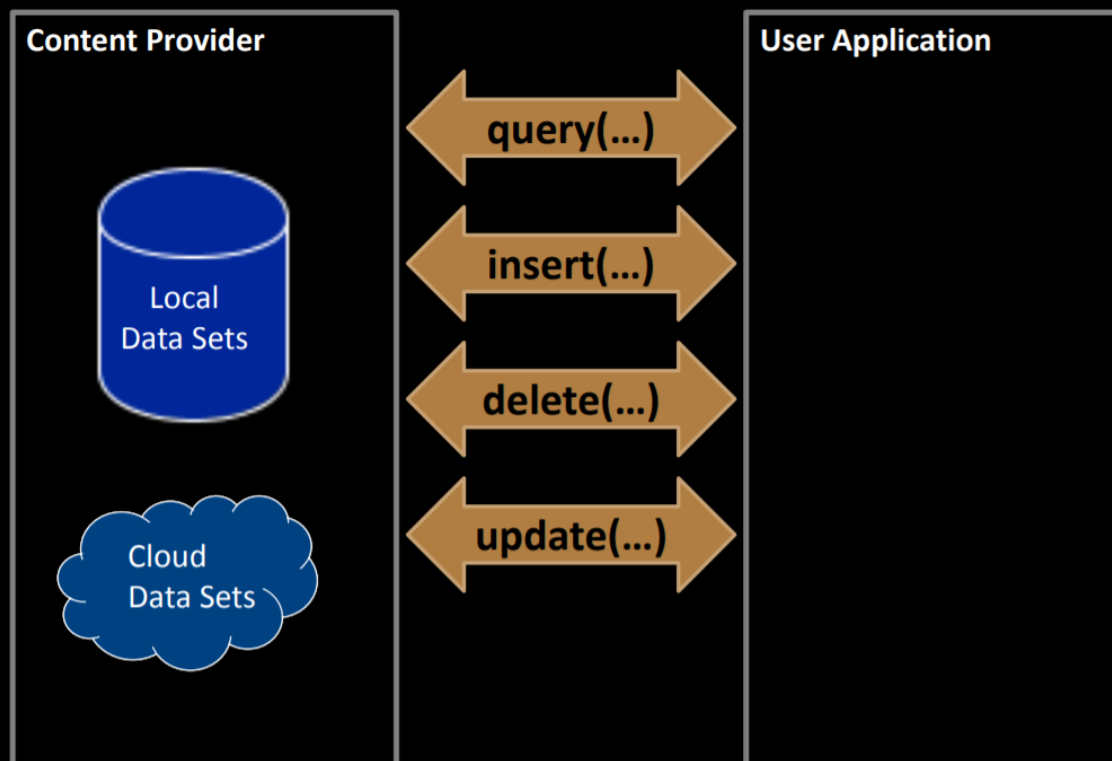


Maps, GPS Navigation

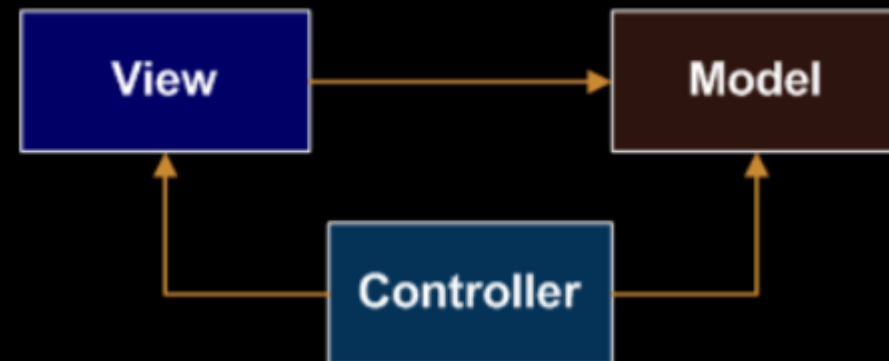
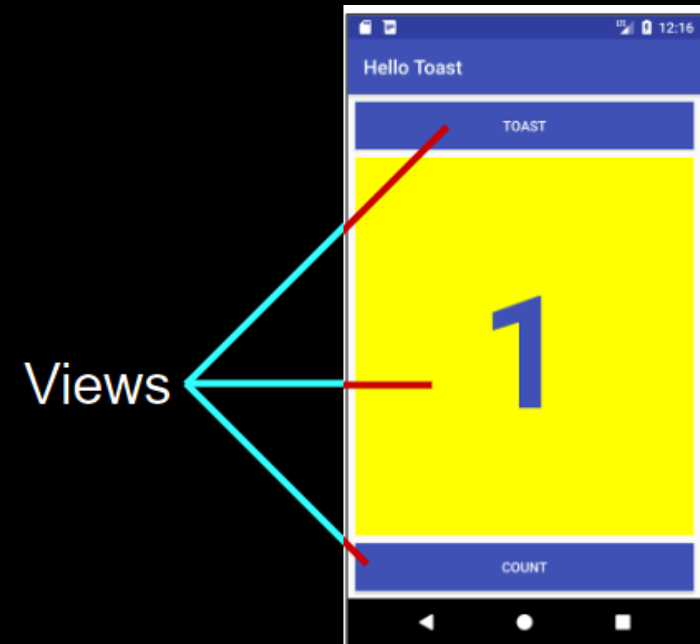
- **BroadcastReceiver Class** – dedykowany *listener* oczekujący na wiadomość zwykle typu *system-wide* w celu wykonania pewnych czynności
- Przykładowo: niski poziom baterii, wi-fi dostępne, ostrzeżenie przed radarami
- Nie posiadają interfejsu
- Rejestrowanie wiadomości typowo po kluczu – jeśli wiadomość odpowiada kluczowi odbiornik jest aktywowany
- Zazwyczaj odpowiada poprzez wykonanie specjalnej czynności lub przekazanie użytkownikowi wiadomości/powiadomienia



- ContentProvider Class – zapewnia dostęp do danych wielu aplikacjom
- Dane globalne: lista kontaktów, zdjęcia, wiadomości, filmy, email
- Dane globalne są zazwyczaj przechowywane w bazie danych (SQLite)
- Klasa oferuje metody innym aplikacjom: retrieve, delete, update, insert
- Wrapper ukrywający właściwe dane. Dostępny interfejs.

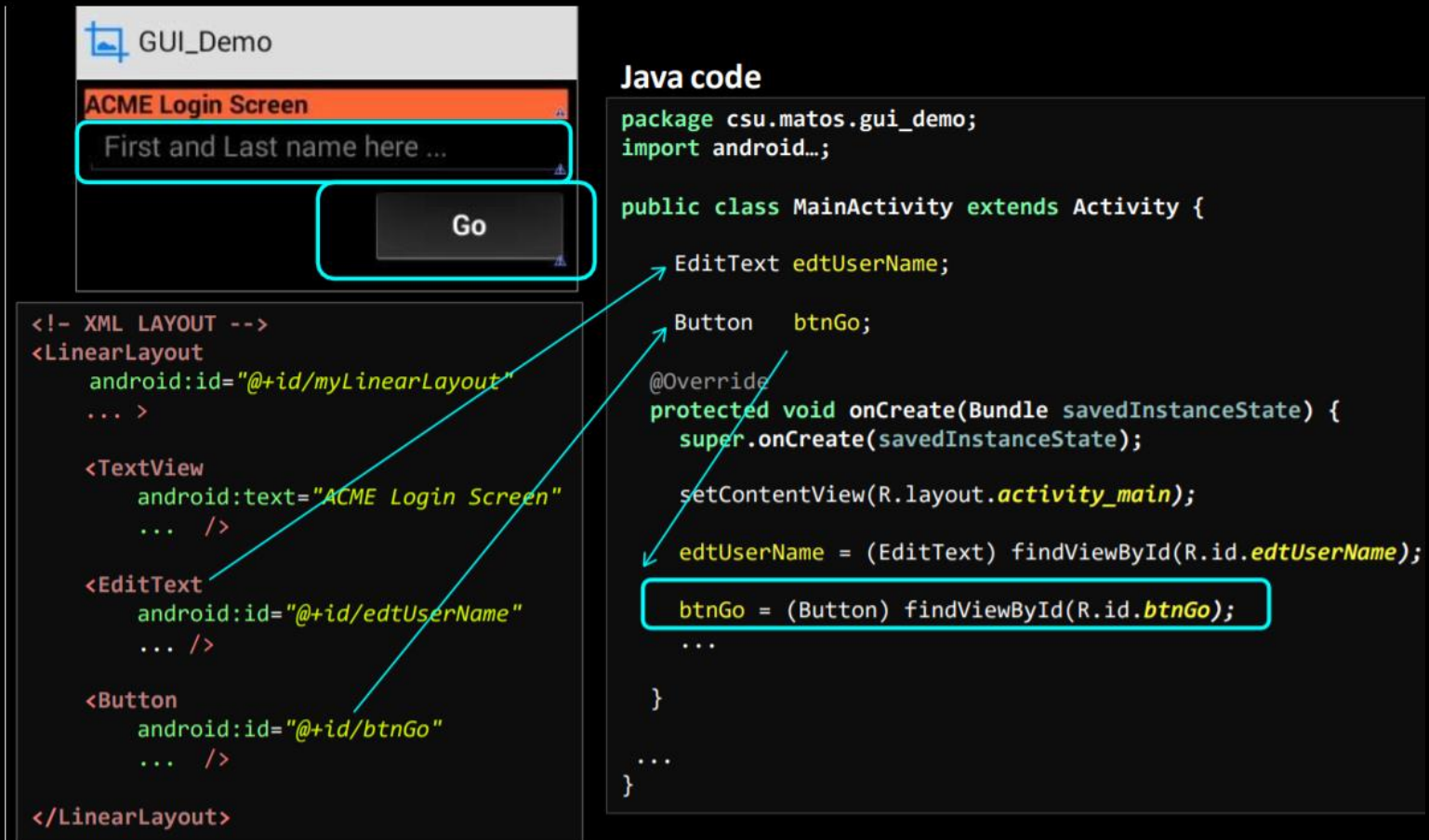


- Wzorzec **Model-Widok-Kontroler (MCV)**
- Jego głównym zadaniem jest odseparowanie interfejsu użytkownika oraz logiki aplikacji
- **Model**
 - Kod w języku JAVA + odwołania do API zarządzające zachowaniem danych aplikacji
- **Widok**
 - Zestaw widoków (ekranów) z którymi użytkownik aplikacji wchodzi w interakcję
- **Kontroler**
 - Interpretacja zdarzeń
 - Źródła zdarzeń: klawiatura, ekran dotykowy, GPS itp



- Jeżeli UI zostało stworzone w plik `activity_main.xml` możemy powiązać układ z zadaną aktywnością wywołując metodę `setContentView(R.layout.activity_main.xml)`
- Poszczególne widżety adresujemy wywołując metodę

`Button btnon = findViewById(R.id.myButton)`



The image shows a visual representation of an Android application's user interface and its underlying code. On the left, a preview of the 'GUI_Demo' app displays a login screen titled 'ACME Login Screen'. It features a text input field labeled 'First and Last name here ...' and a 'Go' button. Red boxes highlight these UI elements. Below the preview is the XML layout code for `activity_main.xml`, which defines a `LinearLayout` containing a `TextView` and an `EditText`. On the right, the Java code for `MainActivity` is shown. It includes the package declaration, imports, and the `onCreate` method. In the `onCreate` method, the `setContentView` call is followed by two lines of view initialization: `edtUserName = (EditText) findViewById(R.id.edtUserName);` and `btnGo = (Button) findViewById(R.id.btnGo);`. Red boxes highlight these two lines of code. Arrows connect the red boxes in the XML code to the corresponding UI elements in the app preview, and another arrow connects the red box around the `btnGo` initialization line to the 'Go' button in the preview.

GUI_Demo

ACME Login Screen

First and Last name here ...

Go

```
<!-- XML LAYOUT -->
<LinearLayout
    android:id="@+id/myLinearLayout"
    ... >

    <TextView
        android:text="ACME Login Screen"
        ... />

    <EditText
        android:id="@+id/edtUserName"
        ... />

    <Button
        android:id="@+id/btnGo"
        ... />

</LinearLayout>
```

Java code

```
package csu.matos.gui_demo;
import android...;

public class MainActivity extends Activity {

    EditText edtUserName;

    Button    btnGo;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

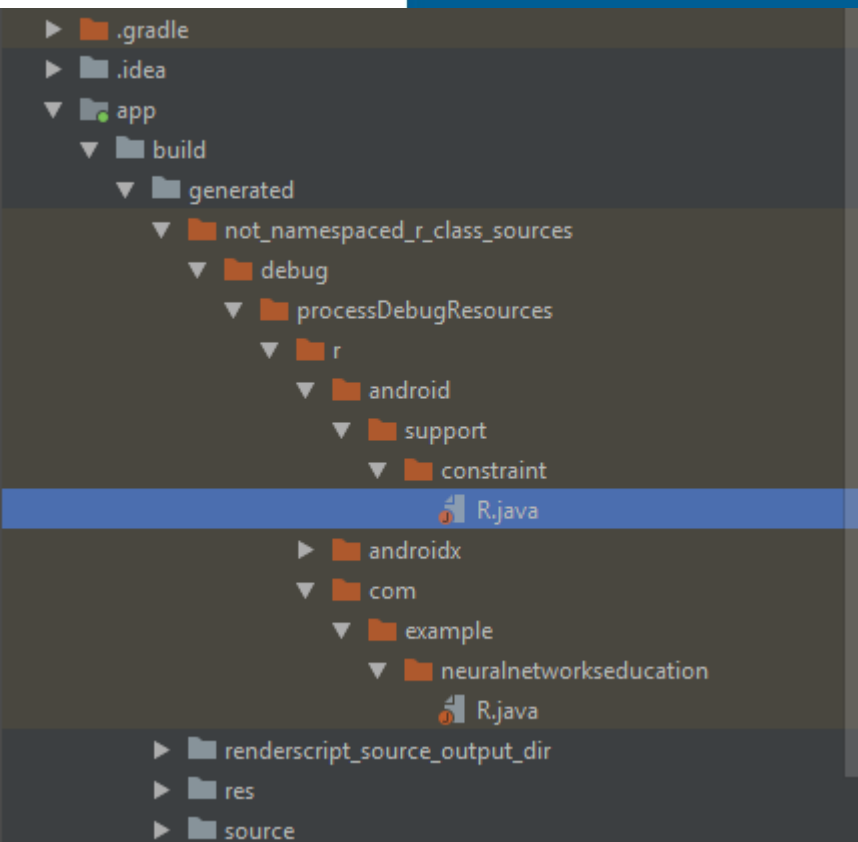
        setContentView(R.layout.activity_main);

        edtUserName = (EditText) findViewById(R.id.edtUserName);

        btnGo = (Button) findViewById(R.id.btnGo);

        ...
    }

    ...
}
```

the build folder are generated and should not be edited.

```

3269     }
3270     public static final class id {
3271         public static final int @+id/menu_tryAtHome=0x7f0d0483;
3272         public static final int DetailFilter=0x7f0d0222;
3273         public static final int LinearLayout1=0x7f0d016a;
3274         public static final int Logout=0x7f0d0215;
3275         public static final int about_dashboard=0x7f0d0257;
3276         public static final int about_text=0x7f0d006e;
3277         public static final int about_us=0x7f0d0213;
3278         public static final int ac_home=0x7f0d043e;

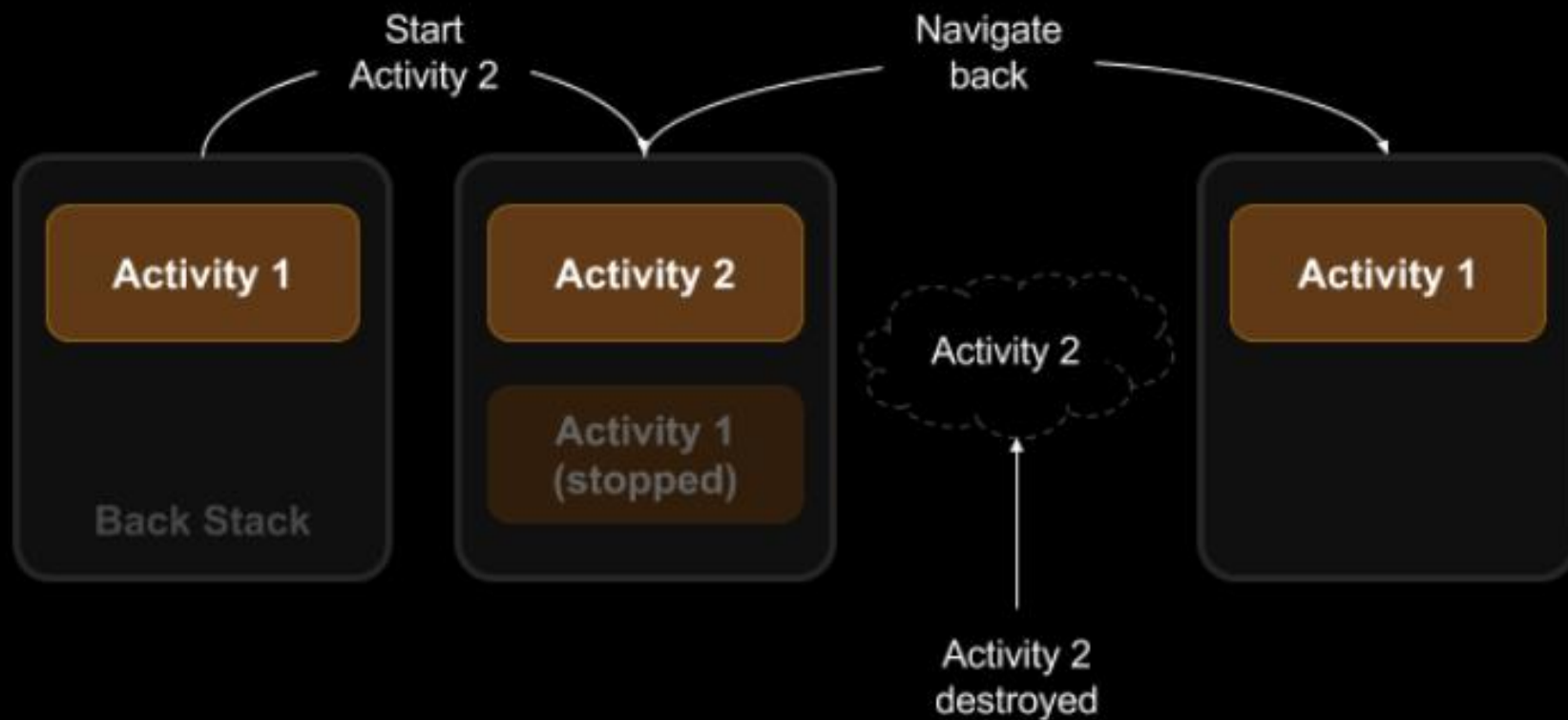
```

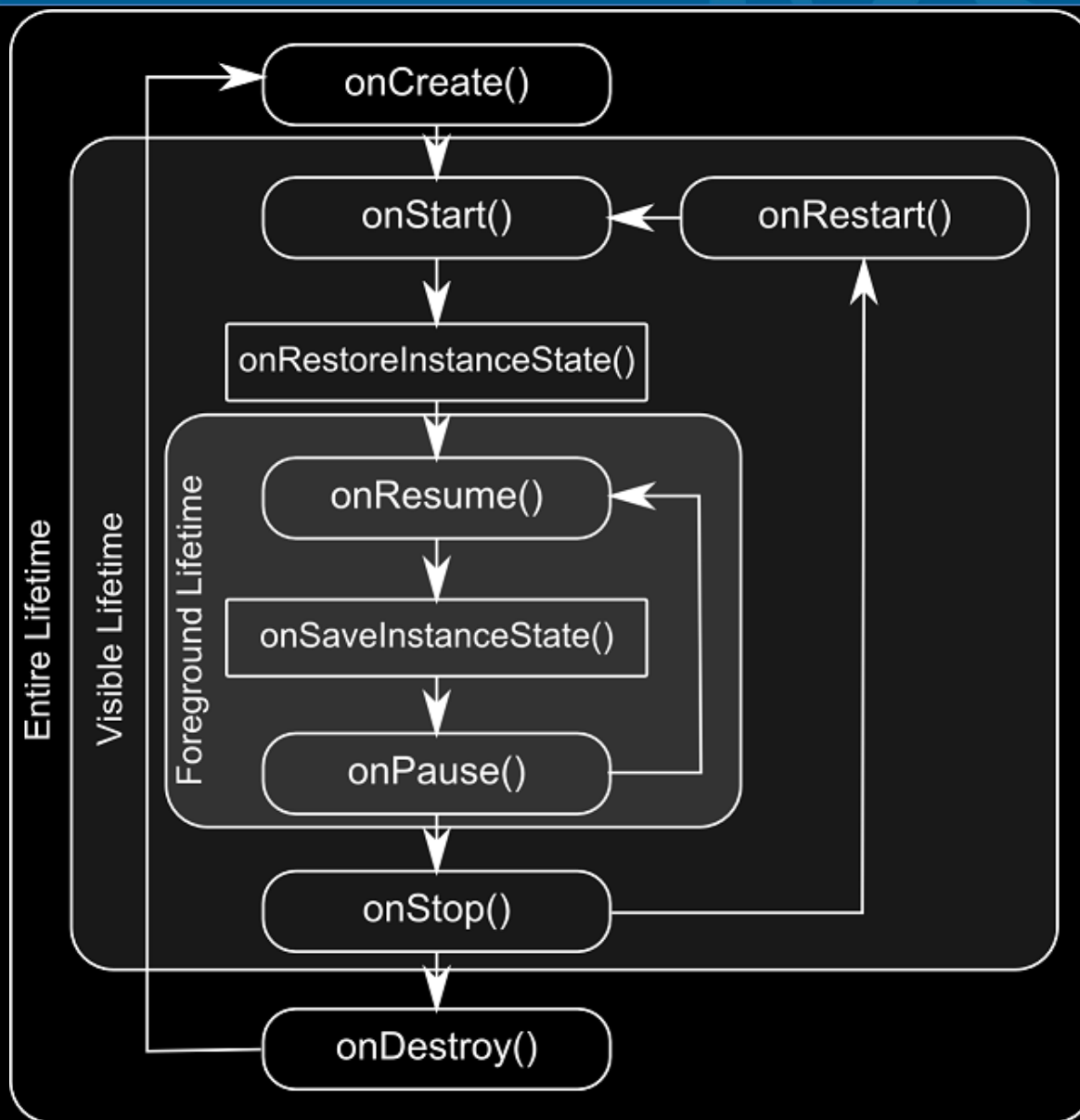
- Każda aplikacja Androida działa w swojej własnej instancji maszyny wirtualnej
- W każdym momencie kilka instancji maszyny wirtualnej może być aktywna (rzeczywista równoległość – nie *task switching*)
- Aplikacja Androida nie kontroluje całkowicie realizacji swojego cyklu życia
- OS może zakończyć każdy proces
 - Zasoby są krytycznie niskie
 - Duża liczba działających aplikacji
 - Aplikacja wymagająca bardzo dużych zasobów (energia, pamięć)

Start

**Life as an Android Application:
Active / Inactive
Visible / Invisible**

End





1. Active/Running

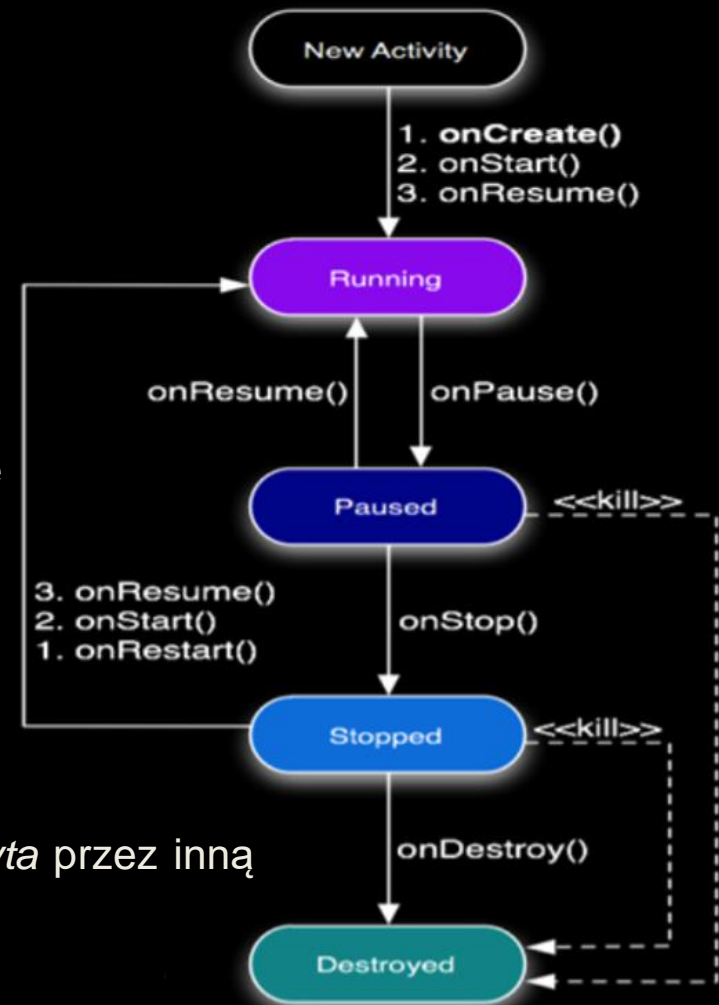
- Aplikacja jest na pierwszym planie
- Znajduje się na szczycie stosu

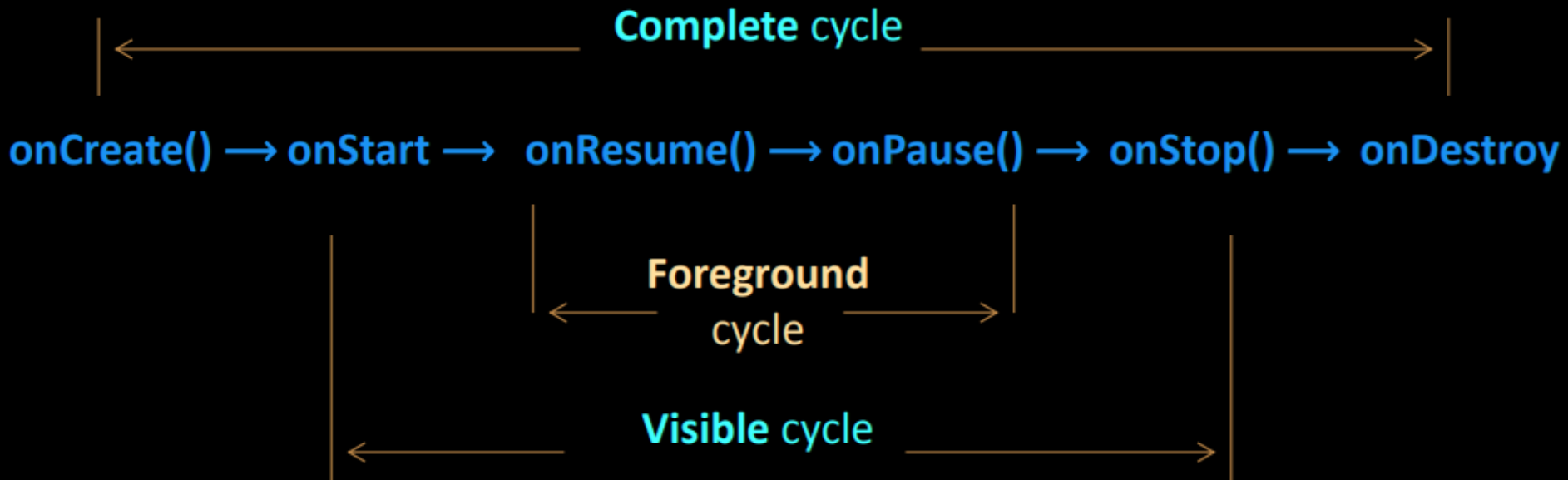
2. Paused

- Ta aktywność dalej pozostaje widoczna dla użytkownika
- Aktywność pozostająca w tym stanie zachowuje swój stan informacji
- Może zostać zabita przez OS
- Kontynuuje *update* UI itp

3. Stopped

- Aktywność zatrzymana jest całkowicie *przykryta* przez inną aktywność
- Nie jest widoczna dla użytkownika
- *Update* UI itp zostaje zatrzymany





- Aplikacja nie musi implementować wszystkich metod przejścia
- Są jednak metody wymagane, oraz wysoko rekomendowane
- Wymagane:
 - `onCreate()` – musi zostać zaimplementowana przez każdą aktywność – początkowe ustawienia
 - Ta metoda jest wywoływana tylko raz podczas jednego cyklu
- Rekomendowane
 - `onPause()` – powinna zostać zaimplementowana gdy aplikacja posiada dane które chcemy zachować

- Pierwsza wywoływana metoda gdy aktywność jest uruchamiana
- Większość kodu znajduje się w tej metodzie
- Typowo używana do zainicjowania struktur danych aplikacji, połączenia elementów UI, zdefiniowania zachowania elementów itp.
- Może otrzymać obiekt *Bundle* zawierający poprzedni stan aktywności
- Metoda poprzedzająca onStart()

@Override

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    // The activity is being created.  
}
```

- Wywoływana w momencie zmiany aktywności
- Używana do zapisania danych i zatrzymania pracy
- Metoda poprzedzająca
 - onResume() – gdy aktywność wraca na pierwszy plan
 - onStop() – gdy aktywność przestaje być widoczna dla użytkownika
- Spauzowana aktywność może zostać zabita przez system

@Override

```
protected void onPause() {  
    super.onPause();  
    // Another activity is taking focus  
    // this activity is about to be "paused"  
}
```

- OS może zabić apke gdy zasoby są niezbędne do operacji o wyższym znaczeniu
- Gdy aktywność osiągnie metody onPause(), onStop(), onDestroy(), może zostać zabita
- onPause() jest jedynym stanem gwarantującym ukończenie przed zabiciem

```
@Override
```

```
protected void onDestroy() {
```

```
    super.onDestroy();
```

```
    // The activity is about to be destroyed.
```

```
}
```

Zmiana konfiguracji wymaga przeładowania layoutu oraz innych zasobów gdy:

- Następuje zmiana orientacji urządzenia
- Zostaje zmieniony język
- Użytkownik wejdzie w tryb multi-window

Przy zmianie konfiguracji Android:

1. Wyłącza aktywność – `onPause()`, `onStop()`, `onDestroy()`
2. Startuje nową instancję aktywności – `onCreate()`, `onStart()`, `onResume()`

