Cykl życia aplikacji i aktywności

PUM

Aplikacja

- Każda aplikacja uruchamiana jest jako należąca do innego użytkownika.
- Tylko użytkownik o określonym ID (właściciel) ma bezpośredni dostęp do wszystkich plików danej aplikacji, w tym plików zapisanych w zasobach i pamięci wewnętrznej urządzenia.
- Aplikacje mogą się komunikować i przekazywać sobie dane poprzez mechanizm Content provider.
- Każda aplikacja jest izolowana od pozostałych – uruchamiana w osobnej instancji wirtualnej maszyny.
- Każda aplikacja to osobny proces systemowy. Proces jest uruchamiany, gdy jakikolwiek komponent aplikacji musi zostać uruchomiony i zakończony, gdy nie jest potrzebny, bądź należy zwolnić zasoby (w wyniku ich zapełnienia).

Elementy składowe aplikacji

- Aplikacja na Androida składa się z jednego lub więcej elementu składowego.
- Takim elementem może być:
  - 1. Activity - aktywność
  - 2. Service - usługa
  - 3. Broadcast receiver – odbiorca (słuchacz) komunikatów
  - 4. Content provider – dostawca treści

Aktywność

- Zwykle aplikacja składa się z jednej lub więcej aktywności.
- Aktywność zazwyczaj utożsamiana jest z pojedynczym ekranem GUI
- Tylko jedna aktywność (zwana główną) jest wybrana do wyświetlania przy pierwszym uruchomieniu aplikacji.
- Aktywność może przekazać sterowanie (i dane) do innej aktywności wykorzystując protokół komunikacyjny zwany intencją (intent).

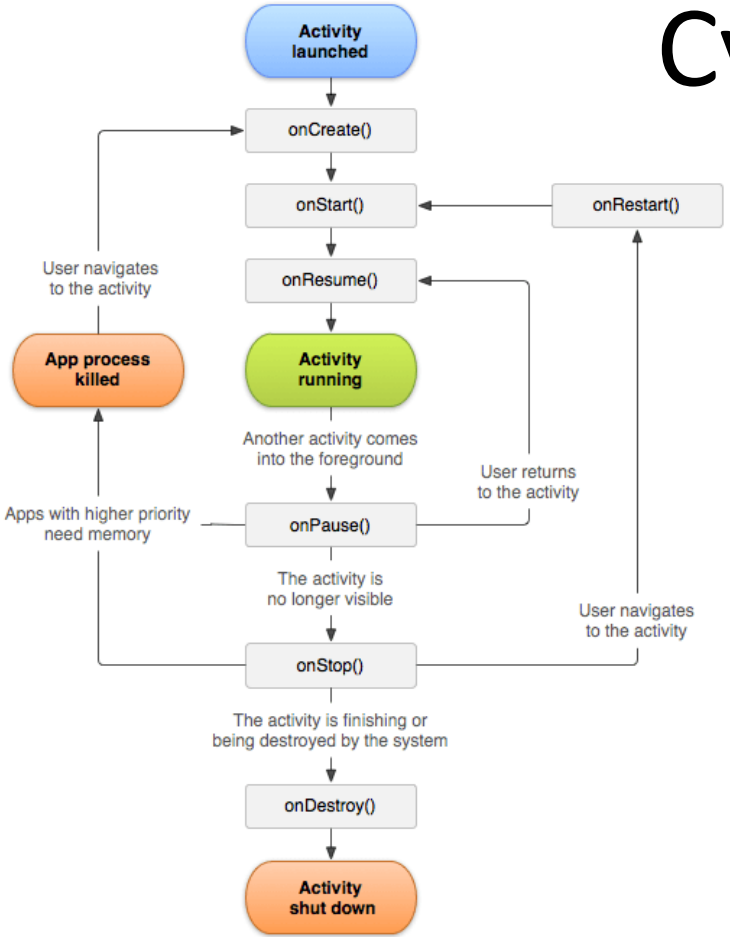
Usługi

- Usługa to specjalny typ aktywności, która nie posiada wizualnej reprezentacji
- Usługi uruchamiane są w tle
- Usługa może być uruchomione na czas nieograniczony, może też być uruchomiona na czas wykonaniu zadania
- Aplikacje mogą uruchamiać własne usługi lub korzystać z już aktywnych
- Przykład: Usługa GPS działa w tle i może co jakiś czas wysyłać dane lokalizacyjne do aplikacji nimi zainteresowanych

Kontekst (Context)

- Dla Androida **`Context`** definiuje tzw. środowisko, w którym aplikacja może tworzyć i używać zasobów.
- Kiedy widżet jest tworzony, jest mu przyporządkowany określony kontekst. Poprzez afiliację do tego środowiska, ma on wówczas dostęp do innych składowych hierarchii, do której został przyporządkowany.
- Dla prostej aplikacji składającej się wyłącznie z jednej aktywności np. MainActivity metoda **`getApplicationContext()`** i referencja **`MainActivity.this`** zwracają ten sam rezultat.
- Aplikacja ma jednak zwykle wiele aktywności. W takiej sytuacji dostępny jest jeden globalny kontekst aplikacji oraz po jednym kontekście dla każdej z aktywności, umożliwiającym dostęp do zasobów zdefiniowanych w ramach tego kontekstu.
- Toast, nowa intencja, kolejka żądań http wymagają podania kontekstu, zwykle wystarczy **`this`**.

Cykl życia aktywności



Koniec życia aktywności

- Metody onPause(), onStop(), onDestroy() mają status killable, czyli po zakończeniu dowolnej z nich, pozostałe nie muszą zostać wywołane (jeśli system wymusi zakończenie aplikacji).
- onPause() to jedyna metoda, która na pewno będzie wywołana przed zakończeniem aplikacji.
- Metoda onPause() powinna być wykorzystywana do zapisania stanu aplikacji.

Cykl życia aktywności

- **`onCreate(Bundle savedInstanceState)`** – wywoływana przy pierwszym uruchomieniu aktywności. Jej parametr zapewnia odtworzenie stanu aktywności, na wypadek gdyby system skasował ją z pamięci. Umieszczamy w niej wszelkie statyczne elementy aplikacji, np. ładujemy widok, ustawiamy Listenery itp.

- **`onResume()`** – wywoływana, gdy aktywność ponownie stanie się procesem na pierwszym planie, mimo że sam ekran aktywności nie musi jeszcze być wyświetlony. Można w niej np. pobrać i rozpocząć odtwarzanie muzyki czy animacji.

- **`onPause()`** – wywoływana w momencie gdy aktywność jest spychana na dalszy plan. W metodzie zatrzymujemy odtwarzanie multimediów, zwalniamy niepotrzebne zasoby, zapisujemy dane (niepotrzebny jest przycisk „Zapisz”).

- **`onDestroy()`** – wywoływana w momencie usuwania aktywności, czyli gdy aktywność została planowo zakończona (**`finish()`**), bądź skasowana przez system. Te przypadki można odróżnić za pomocą metody **`isFinishing()`**, która zwróci **`false`** w tym drugim przypadku.
- W metodzie niszczymy to, co się da ;), tzn. usuwamy statyczne dane.

- **UWAGA!** Metody **`onDestroy()`** i **`onStop()`** mogą się nigdy nie wykonać, gdy aplikacja zostanie skasowana zaraz po wywołaniu **`onPause()`**.

Badanie cyklu życia aktywności - komunikaty

1. Drukowanie komunikatów w dzienniku:

```
Log. println(int priority, String tag, String msg);
```

Priority: ASSERT, DEBUG, ERROR, INFO, VERBOSE, WORN

tag – identyfikator źródła komunikatu (dowolny tekst)

msg - komunikat

```
Log.println(INFO, "Cykl zycia", "onStart");
```

2. Tosty (Toasty)

```
Toast.makeText(getApplicationContext(),"onStart",Toast.LENGTH_LONG).show();
```

3. Wysyłanie komunikatów przez internet

Drukowanie komunikatów w dzienniku

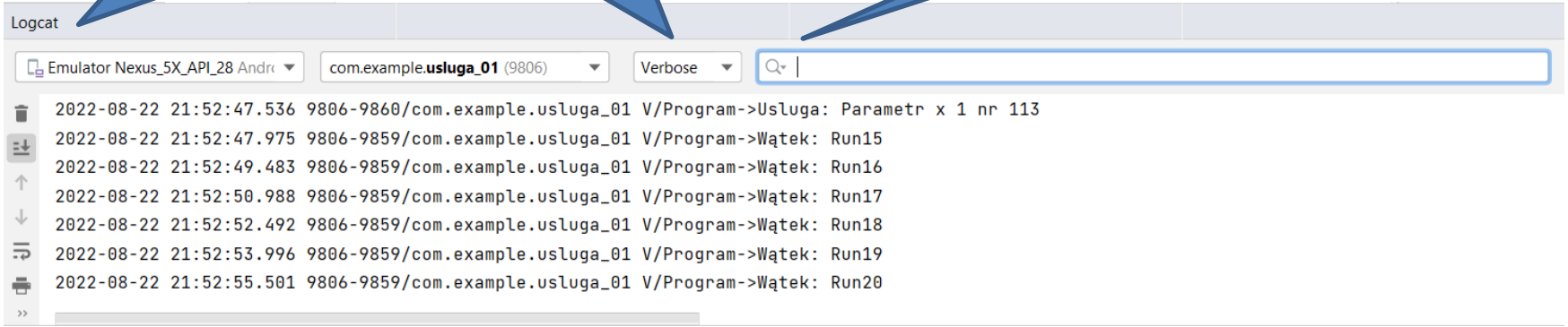
Uproszczona wersja wpisów do dziennika:

```
Log.v(String, String);    //verbose
Log.d(String , String);   //debug
Log.i(String , String);   //information
Log.w(String , String);   //warn
Log.e(String , String);   //error
```

Dziennik Logcat wyświetlany u dołu ekranu

Wybór priorytetu komunikatów

Filtr komunikatów



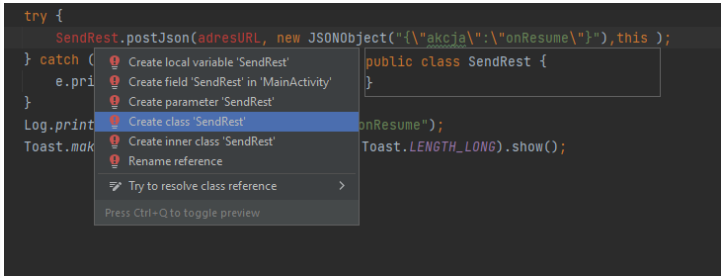
Odbiornik komunikatów http

http://lukan.sytes.net:1880/#flow/6c4226e0.4a4a78

Fragment kodu – trzy rodzaje komunikatów

```java
@Override
protected void onResume() {
    super.onResume();
    try {
        SendRest.postJson(adresURL, new JSONObject("{\"akcja\":\"onResume\"}"),this );
    } catch (JSONException e) {
        e.printStackTrace();
    }
    Log.println(INFO, "Cykl zycia", "onResume");
    Toast.makeText(this,"onResume",Toast.LENGTH_LONG).show();
}

@Override
protected void onRestart() {
    super.onRestart();
    try {
        SendRest.postJson(adresURL, new JSONObject("{\"akcja\":\"onRestart\"}"),this );
    } catch (JSONException e) {
        e.printStackTrace();
    }
    Log.println(INFO, "Cykl zycia", "onRestart");
    Toast.makeText(this,"onResume",Toast.LENGTH_LONG).show();
}
```

SendRest – nowa klasa, którą należy utworzyć

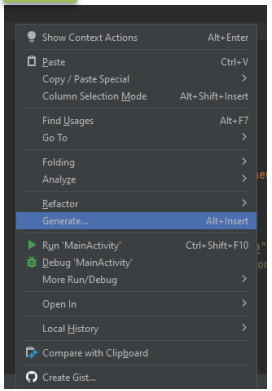



```java
String adresURL = "http://lukan.sytes.net:1880/pum/cykl";
```

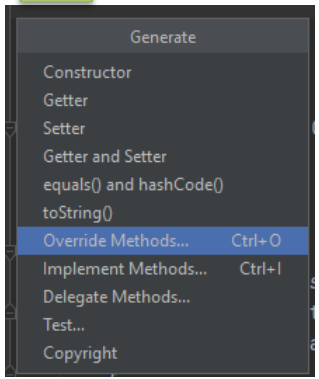
Dodawanie metod @Override

1. Kliknięcie prawym klawiszem w kodzie klasy

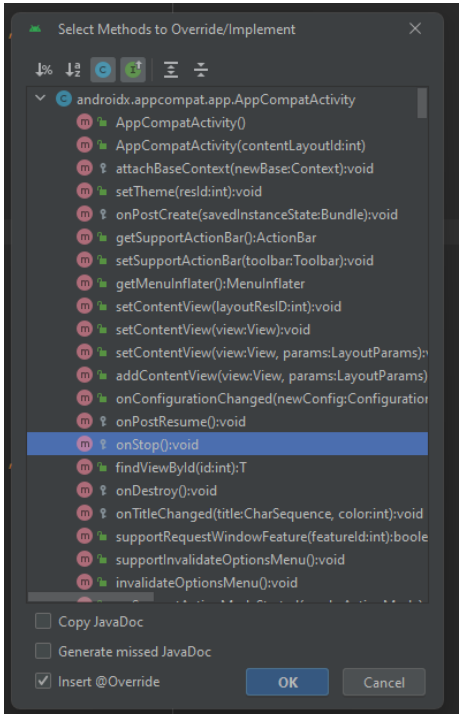
2.



3.



4.

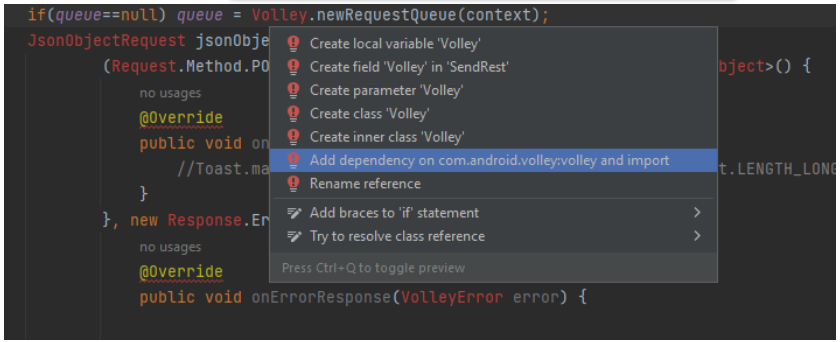


Kod klasy SendRest

wysyłanie żądań http - metoda postJson()

```
public class SendRest {

    public static RequestQueue queue;

    public static void postJson(String url, JSONObject jsonObject, Context context){
        if(queue==null)  queue = Volley.newRequestQueue(context);

        JsonObjectRequest jsonObjectRequest = new JsonObjectRequest
                (Request.Method.POST, url, jsonObject, new Response.Listener<JSONObject>() {
                    @Override
                    public void onResponse(JSONObject response) {
                      //Toast.makeText(context,"--->" + response.toString(),Toast.LENGTH_LONG).show();
                    }
                }, new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError error) {

                    }
                });

        queue.add(jsonObjectRequest);
    }
}
```

Dodanie biblioteki (zależności)



Na podstawie: https://developer.android.com/training/volley/request

Zezwolenie na dostęp do internetu

Od API 28 połączenia internetowe domyślnie tylko HTTPS

Aby zezwolić na połączenia niebezpieczne HTTP w pliku manifestu: $project_path\android\app\src\debug\AndroidManifest.xml

należy dokonać małego wpisu <application android:usesCleartextTraffic="true" />

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

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

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

Zezwolenie na dostęp do internetu

Zezwolenie na połączenia http

Plik APK

- app/build/outputs/aps/debag/app-debag.apk

Intencje

- Najprostsze aplikacje Java są jednowątkowe. Program wywołuje metody w sposób synchroniczny, czekając na wynik.
- Aplikacje działające pod kontrolą systemu Android mogą składać się z wielu niezależnych ale pozostających w kooperacji aktywności. Każda aktywność działa w ramach swojego wątku, a jedna z nich określana jest mianem aktywności głównej.
- Android wykorzystuje metodę **startActivity(Intent)** by uruchomić aktywność, która staje się aktywna i potencjalnie widoczna dla użytkownika, jednakże obiekt wywołujący cały czas znajduje się w obrębie swojego wątku.
- By aktywność nadrzędna uruchomiła aktywność podrzędną oraz pozyskała rezultat jej działania, wykorzystywana jest metoda:

  startActivityForResult (Intent, requestCodeID)

  Gdzie requestCodeID to arbitralnie dobierana wartość identyfikująca obiekt wywołujący (podobnie jak alias).
- Rezultat wykonania zwrócony przez aktywność podrzędną (jeżeli jest dostępny) może zostać przekazany w sposób asynchroniczny poprzez wywołanie zwrotne typu:

  onActivityResult (requestCodeID, resultCode, Intent)

Otwarcie nowej aktywności

Otwarcie nowej aktywności

```
public void onClickButton0(View view) {
   Intent myActivity = new Intent(MainActivity.this, MainActivity2.class);
   startActivity(myActivity);
}
```

Otwarcie nowej (podrzędnej) aktywności i przekazanie jej parametru extra

```
Intent myActivity = new Intent(MainActivity.this, MainActivity2.class);
myActivity.putExtra("tekst", "Ala ma kota");
startActivity(myActivity);
```

Pobranie wartości przesłanego parametru w nowej aktywności

```
TextView textView1 = findViewById(R.id.textView);
textView1.setText(getIntent().getStringExtra("tekst").toString());
```

Aby otworzyć aktywność trzeba ją najpierw dodać do projektu z Menu:

File/New/Activity/Empty Views Activity

Oczywiście do projektu można dodać inną niż pustą aktywność

Otwarcie nowej aktywności

w celu uzyskania od niej danych

Aktywność 1. otwiera aktywność 2. i przekazuje jej dane

```
int mojKod = 117;
public void onClick(View view) {
    Intent intent = new Intent(MainActivity.this, MainActivity2.class);
    intent.putExtra("MSG",((TextView)findViewById(R.id.editTextWiadomosc)).getText().toString());
    startActivityForResult(intent, mojKod );
}
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == mojKod) {
        if (resultCode == RESULT_OK) {
            String response = data.getStringExtra("RESPONSE");
            Toast.makeText(this, "Witaj " + response,
                    Toast.LENGTH_SHORT).show();
            ((TextView)findViewById(R.id.textViewOdpowiedz)).setText(response);
        }
    }
}
```

Podejście przestarzałe – Deprecated: https://developer.android.com/training/basics/intents/result

Otwarcie nowej aktywności

w celu uzyskania od niej danych

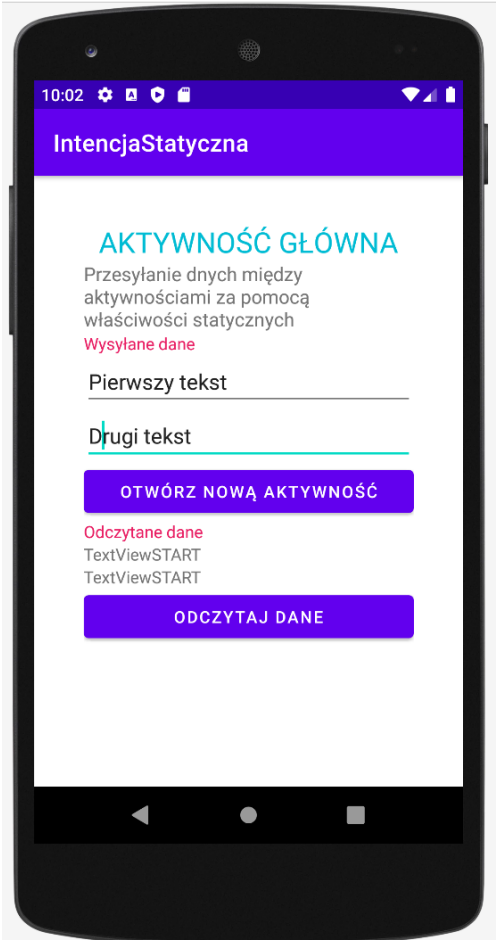
Aktywność 2. otwarta przez aktywność 1. zwraca dane do aktywności 1 i jest zamykana.

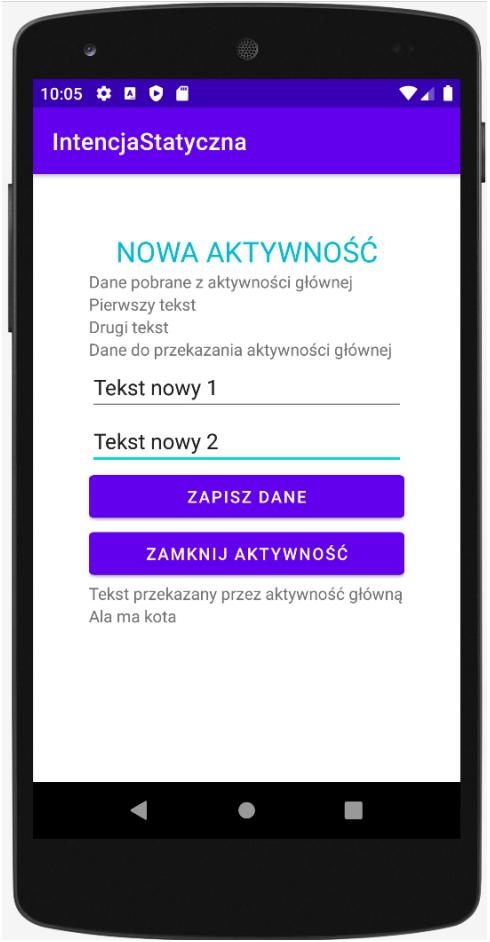
```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);

    ((TextView)findViewById(R.id.textView2)).setText(getIntent().getStringExtra("MSG"));
    Toast.makeText(this, getIntent().getStringExtra("MSG"),
            Toast.LENGTH_SHORT).show();
}

public void onClick(View view) {
    Intent intent = new Intent();
    intent.putExtra("RESPONSE", ((TextView)findViewById(R.id.editTextODP)).getText().toString());
    setResult(RESULT_OK, intent);
    finish();
}
```

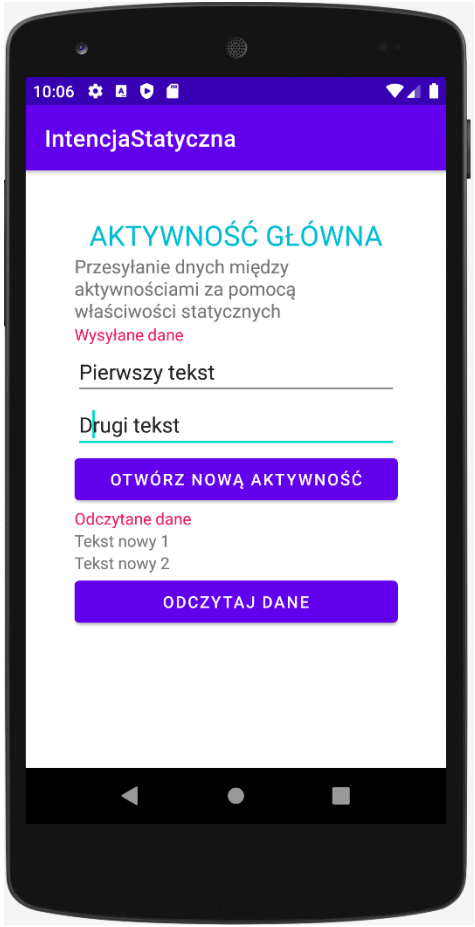
Bardzo prosto można zwrócić dane do aktywności 1 przez właściwości statyczne

Właściwości statyczne

- Aktywności mogą komunikować się poprzez właściwości statyczne
- Aktywności wzajemnie widzą i mogą czytać oraz zmieniać swoje i cudze właściwości statyczne


10:02
IntencjaStatyczna
AKTYWNOŚĆ GŁÓWNA
Przesyłanie dnych między aktywnościami za pomocą właściwości statycznych
Wysyłane dane
Pierwszy tekst
Drugi tekst
OTWÓRZ NOWĄ AKTYWNOŚĆ
Odczytane dane
TextViewSTART
TextViewSTART
ODCZYTAJ DANE


10:05
IntencjaStatyczna
NOWA AKTYWNOŚĆ
Dane pobrane z aktywności głównej
Pierwszy tekst
Drugi tekst
Dane do przekazania aktywności głównej
Tekst nowy 1
Tekst nowy 2
ZAPISZ DANE
ZAMKNIJ AKTYWNOŚĆ
Tekst przekazany przez aktywność główną
Ala ma kota


10:06
IntencjaStatyczna
AKTYWNOŚĆ GŁÓWNA
Przesyłanie dnych między aktywnościami za pomocą właściwości statycznych
Wysyłane dane
Pierwszy tekst
Drugi tekst
OTWÓRZ NOWĄ AKTYWNOŚĆ
Odczytane dane
Tekst nowy 1
Tekst nowy 2
ODCZYTAJ DANE

Aktywność główna

```
public class MainActivity extends AppCompatActivity {

    static String text1;
    static String text2;
    static String text3 = "brak";
    static String text4 = "brak";

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

        ((Button)findViewById(R.id.button)).setOnClickListener(view -> {
            text1 = ((TextView)findViewById(R.id.editText1)).getText().toString();
            text2 = ((TextView)findViewById(R.id.editText2)).getText().toString();
            Intent myActivity = new Intent(MainActivity.this, MainActivity2.class);
            startActivity(myActivity);
        });

        ((Button)findViewById(R.id.button3)).setOnClickListener(view -> {
            ((TextView)findViewById(R.id.textView4)).setText(MainActivity.text3);
            ((TextView)findViewById(R.id.textView5)).setText(MainActivity.text4);
        });
        MainActivity2.text13 = "Ala ma kota";
    }
}
```

Nowa aktywność otworzona przez główną

```
public class MainActivity2 extends AppCompatActivity {

    static String text13;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);

        ((TextView)findViewById(R.id.textView1)).setText(MainActivity.text1);
        ((TextView)findViewById(R.id.textView2)).setText(MainActivity.text2);

        ((Button)findViewById(R.id.button2)).setOnClickListener(view -> {
            MainActivity.text3 = ((TextView)findViewById(R.id.editText11))
                    .getText().toString();;
            MainActivity.text4 = ((TextView)findViewById(R.id.editText12))
                    .getText().toString();;
        });

        ((Button)findViewById(R.id.button4)).setOnClickListener(view -> finish());

        ((TextView)findViewById(R.id.textView9)).setText(text13);
    }
}
```

Zagadnienia

- Komunikacja aplikacji z użytkownikiem
- Komunikacja między aktywnościami
- Komunikacja aplikacji z innymi aplikacjami – intencjami
- Komunikacja z lokalną bazą danych SQLite
- Komunikacja internetowa
  - http – korzystanie z usług REST i SOAP
  - MQTT – Internet rzeczy, komunikatory, przesyłanie wiadomości między urządzeniami
- Komunikacja z usługami i czujnikami