



Programowanie dla Androida

Ubiquitous

23 marca 2020 r.

Fragmenty

- To w pełni niezależny fragment interfejsu użytkownika wraz z logiką, który można umieścić w aktywności
- Mogą zostać użyte tylko wewnątrz aktywności
- Na fragment składa się pliku XML opisujący wygląd i klasa opisująca zachowanie
- Dostępne od wersji 3 Android SDK

Intencje

- To mechanizm umożliwiający wywoływanie i komunikację pomiędzy aktywnościami
- Intencja umożliwia wywołanie innej aktywności aplikacji lub aktywności zarejestrowanej w systemie
- Umożliwiają też komunikację z usługami (Service) i odbiorcami komunikatów (Broadcast receivers)

Intencje explicite

- Intencje wprost odwołują się do konkretnej aktywności, podając jej dane (nazwę klasy)
- Najczęściej są używane do wywołania aktywności wewnątrz aplikacji
- Dane można przekazać przez obiekt klasy Intent, korzystając z metody putExtra()
- Dane są przesyłane jako pary (klucz, wartość)

Intencje explicite

- Przydatny import

```
import android.content.Intent
```

- Wywołanie intencji

```
fun showActivity() {  
    val i=Intent( packageContext: this,SecondActivity::class.java)  
    i.putExtra( name: "myString", value: "This is a message for ActivityB")  
    i.putExtra( name: "myInt", value: 100)  
    startActivity(i)  
}
```

Intencje

- Aby odczytać przekazane dane, klasa Activity ma własność intent, zwracającą wywołującą ją intencję
- W klasie Intent jest własność extras, zwracająca obiekt zawierające wszystkie przekazane dane
- Z niego, za pomocą metod get... można odczytać dane

```
val extras = intent.extras ?: return  
  
val myString = extras.getString( key: "myString")  
val myInt = extras.getInt( key: "MyInt")
```

Intencje

- Aby aktywność można było wywołać, jej opis musi się znaleźć w pliku manifest

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

    <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/AppTheme">
        <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"></activity>
    </application>

</manifest>
```

Intencje

- W powyższy sposób można przekazać dane tylko do aktywności wywoływanej
- Aby przesłać je też w drugą stronę, aktywność musi być wywołana jako pod-aktywność za pomocą metody `startActivityResult(...)`
- Aktywność wywołana musi przeciążyć metodę `finish()`
- Aktywność wywołująca musi przeciążyć metodę `onActivityResult()`

Intencje

- Aktywność nadrzędna

```
class MainActivity : AppCompatActivity() {  
    val REQUEST_CODE=1000
```

```
fun showActivity() {  
    val i=Intent( packageContext: this,SecondActivity::class.java)  
    startActivityForResult(i,REQUEST_CODE)  
}
```

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
    if ((requestCode==REQUEST_CODE)  
        && (resultCode== Activity.RESULT_OK)) {  
        if (data!=null) {  
            if (data.hasExtra( name: "returnString1")) {  
                statusText.text = data.extras.getString( key: "returnString1")  
            }  
        }  
    }  
}
```

Intencje

- Aktywność podrzędna

```
class SecondActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_second)  
    }  
  
    override fun finish() {  
        val data = Intent()  
  
        data.putExtra(name: "returnString1", value: "Message to parent")  
        setResult(Activity.RESULT_OK, data)  
  
        super.finish()  
    }  
}
```

Intencje implicite

- Intencje nie-wprost nie definiują precyzyjnie klasy ale akcję i dane dla niej
- Np. akcja ACTION_VIEW w połączeniu z adresem URL otworzy aktywność przeglądarki

```
fun showImplicitActivity() {  
    val i = Intent(Intent.ACTION_VIEW, Uri.parse("http://www.predki.com"))  
    startActivity(i)  
}
```

Filtry intencji

- Za ich pomocą aktywność może zgłosić systemowi chęć obsługi wybranych akcji
- Konfiguruje się je w manifeście
- Aktywność musi też uzyskać odpowiednie uprawnienia dla akcji

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ebookfreny.WebView"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="10" />

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

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".WebViewActivity" >
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:scheme="http" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Intencje nadawcze

- Aktywności mogą rozsyłać komunikaty do wszystkich zainstalowanych aplikacji
- Jeżeli aktywność chce otrzymywać komunikaty, musi się zarejestrować
- Do odbioru komunikatów musimy stworzyć klasę dziedziczącą po klasie BroadcastReceiver

Intencje nadawcze

- Są trzy rodzaje intencji nadawczych:
 - zwykłe - po wysłaniu znikają
 - stałe (sticky) - można je odczytać też później
 - dwustronne (ordered) - umożliwiają przesłani danych do nadawcy komunikatu

Wątki

- Główny wątek aplikacji nie powinien wykonywać intensywnych obliczeniowo zadań
- Tylko główny wątek powinien wprost wchodzić w interakcję z interfejsem użytkownika
- Do tworzenie osobnego wątku można wykorzystać klasę AsyncTask
- Klasa AsyncTask jest klasą generyczną, zależną od 3 typów

```
private inner class MyTask : AsyncTask<Type 1, Type 2, Type 3>() {
```

Wątki

- Te 3 typy odpowiadają argumentom metod:
 - doInBackground
 - onProgressUpdate
 - onPostExecute
- Można użyć Void

```
private inner class MyTask : AsyncTask<String, Int, String>() {  
    override fun onPreExecute() {  
          
    }  
  
    override fun doInBackground(vararg params: String): String {  
          
    }  
  
    override fun onProgressUpdate(vararg values: Int?) {  
          
    }  
  
    override fun onPostExecute(result: String) {  
          
    }  
}
```

Wątki

- Metoda `onPreExecute()` jest uruchamiana na początku w głównym wątku
- Główny kod nowego wątku powinien znajdować się w metodzie `doInBackground()`
- Metoda `onProgressUpdate()` działa w głównym wątku i ma dostęp do UI
- Jest uruchamiana w momencie wywołania metody `publishProgress()` w asynchronicznym wątku
- Metoda `onPostExecute()` jest uruchamiana, gdy zakończy się działanie kodu z `doInBackground()` - działa w głównym wątku

Wątki

- W ten sposób może działać jeden wątek w tle
- Kolejne są kolejgowane i uruchamiane po zakończeniu aktywnego wątku
- Można uruchomić wątki równolegle wywołując metodę `executeOnExecutor()` z klasy `AsyncTask`
- O tym, ile wątków może działać równolegle decyduje system w oparciu o liczbę rdzeni procesora

Usługi (Services)

- Długotrwałe operacje powinny być uruchamiane jako usługi
- Usługi uruchamiane są poprzez mechanizm intencji i są niezależne od uruchamiającej je aktywności
- Metody: `startService()`, `stopService()`, `stopSelf()`

Przechowywanie danych

- Aplikacja na Androidzie może zapisywać i odczytywać pliki
- Pliki mogą być prywatne dla aplikacji lub dostępne publicznie
- Można je zapisywać na urządzeniu lub na karcie pamięci

```
package com.predki.fileaccess

class Contact {
    var name: String? = null
    var phone: String? = null

    constructor() {}

    constructor(name:String?, phone:String?) {
        this.name=name
        this.phone=phone
    }

    constructor(line:String?) {
        if (line!=null) {
            val tokens=line.split( ...delimiters: ";")
            if (tokens.size==2) {
                name = tokens[0]
                phone = tokens[1]
            }
        }
    }

    override fun toString(): String {
        return "Contact $name @ $phone"
    }

    fun toCSV():String {
        return "$name;$phone\n"
    }
}
```

Zapis pliku

```
fun saveClick(v:View) {  
    val contacts = Arrays.asList(  
        contact( name: "Darth Vader",  phone: "555328431"),  
        contact( name: "Luke Skywalker",  phone: "5558946766"))  
  
    val filename = "contacts.csv"  
    val file = OutputStreamWriter(openFileOutput(filename, Context.MODE_PRIVATE))  
  
    for (c in contacts) {  
        file.write(c.toCSV())  
    }  
    file.flush()  
    file.close()  
    Toast.makeText( context: this,  text: "Plik został zapisany", Toast.LENGTH_LONG).show()  
}
```

▼	com.predki.fileaccess	d 2018-01-04 10:4	4 KB
▶	cache	d 2018-03-13 11:0	4 KB
▼	files	d 2018-03-13 12:4	4 KB
	contacts.csv	-r 2018-03-15 09:4	48 B

```
Darth Vader;555328431  
Luke Skywalker;5558946766
```

Odczyt pliku

```
fun readClick(v: View) {  
    val contacts = ArrayList<contact>()  
    try {  
        val filename = "contacts.csv"  
        if (FileExists(filename)) {  
            val file = InputStreamReader(openFileInput(filename))  
            val br = BufferedReader(file)  
  
            var line = br.readLine()  
            while (line != null) {  
                contacts.add(contact(line))  
                line = br.readLine()  
            }  
            file.close()  
            val count = contacts.size  
            Toast.makeText(context: this, text: "Wczytano $count kontaktów", Toast.LENGTH_LONG).show()  
        } else {  
            Toast.makeText(context: this, text: "Nie znaleziono pliku", Toast.LENGTH_LONG).show()  
        }  
    } catch (e: Exception) {  
    }  
}
```

```
fun FileExists(fname: String): Boolean {  
    val file = baseContext.getFileStreamPath(fname)  
    return file.exists()  
}
```

Zapis na kartę pamięci

- Aplikacja musi posiadać uprawnienia w manifeście

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

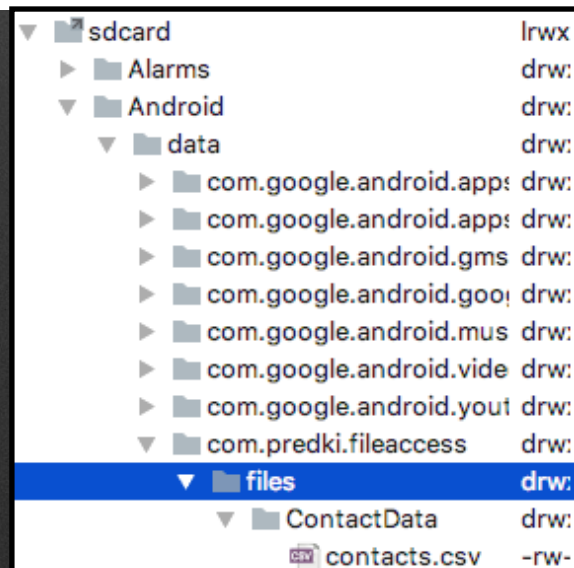
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="FileAccess"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

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

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>
```

Zapis na kartę pamięci

```
fun saveClick(v:View) {  
    val contacts = Arrays.asList(  
        contact( name: "Darth Vader", phone: "555328431"),  
        contact( name: "Luke Skywalker", phone: "5558946766"))  
  
    val filename = "contacts.csv"  
    val path = this.getExternalFilesDir( type: null)  
  
    val contactDirectory = File(path, child: "ContactData")  
    contactDirectory.mkdirs()  
    val file=File(contactDirectory,filename)  
  
    for (c in contacts) file.appendText(c.toCSV())  
  
    Toast.makeText( context: this, text: "Plik został zapisany", Toast.LENGTH_LONG).show()  
}
```



Odczyt z karty pamięci

```
fun readClick(v: View) {  
    val contacts = ArrayList<contact>()  
    try {  
        val filename = "contacts.csv"  
        val path = this.getExternalFilesDir( type: null)  
        val contactDirectory = File(path, child: "ContactData")  
  
        if (contactDirectory.exists()) {  
            val file=File(contactDirectory,filename)  
  
            if (file.exists()) {  
                val br = BufferedReader(file.reader())  
  
                var line = br.readLine()  
                while (line != null) {  
                    contacts.add(contact(line))  
                    line = br.readLine()  
                }  
                val count = contacts.size  
                Toast.makeText( context: this, text: "Wczytano $count kontaktów", Toast.LENGTH_LONG).show()  
            } else {  
                Toast.makeText( context: this, text: "Nie znaleziono pliku", Toast.LENGTH_LONG).show()  
            }  
        }  
  
    } catch (e: Exception) {  
    }  
}
```

Bazy danych



- Domyślną bazą danych na Androidzie jest SQLite
- SQLite został napisany w 2000 r. Przez D. Richard'a Hipp'a w ramach kontraktu dla US Navy
- Kod został udostępniony na zasadzie Public Domain
- Jest to biblioteka napisana w C
- Realizuje większość funkcji standardu SQL-92
- Najpopularniejszy DBMS na świecie
- www.sqlite.org. <http://sqlitebrowser.org>



SQLite

- Nie ma osobnego serwera, kod jest linkowany z aplikacją
- Metody do obsługi bazy danych są w Androidzie opakowane w klasy

SQLite - Cursor

- Klasa służąca do przeglądania wyników zapytania:
 - `close()` - zamyka obiekt
 - `getCount()` - podaje liczbę wierszy
 - `moveToFirst()` - skacze na początek
 - `moveToLast()` - skacze na koniec
 - `moveToNext()` - przesuwają się do następnego wiersza
 - `move(...)` - skacze o zadaną liczbę wierszy
 - `get<typ>(...)` - zwraca wartość z podanej kolumny jako podany typ (wersje `getInt()`, `getFloat()`, ...)

SQLite - SQLiteDatabase

- Klasa zarządza całą bazą danych
 - insert() - wstawia nowy wiersz
 - delete() - usuwa wiersze z tabeli
 - query() - uruchamia zapytanie i zwraca wyniki jako obiekt klasy Cursor
 - execSQL() - umożliwia wywołanie polecenia SQL
 - rawQuery() - działa podobnie jak query()

SQLite - SQLiteOpenHelper

- Klasa zawierająca metody pomagające w zarządzaniu bazą:
 - onCreate() - uruchamiana, gdy baza jest tworzona po raz pierwszy
 - onUpgrade() - uruchamiana, gdy aplikacja zawiera nowszą wersję bazy
 - onOpen() - uruchamiana w momencie otwarcia połączenia z bazą
 - getReadableDatabase() - otwiera bazę tylko do odczytu
 - getWritableDatabase() - otwiera bazę do odczytu/zapisu
 - close() - zamyka połączenie z bazą

Dostawcy treści

- Domyślnie baza danych jest prywatne dla aplikacji
- Aplikację mogą sobie udostępniać dane korzystając z mechanizmu dostawców treści (Content Providers)
- Przykładem może być aplikacja Kontakty
- Dane są udostępniane w formie tabelarycznej (jak baza danych) lub jako łączy do pliku

Metody dostawcy treści

- onCreate() - uruchamiana w momencie inicalizacji
- query() - zewnętrzna aplikacja wywołuje tę metodę aby uzyskać dane
- insert() - jak wyżej, tylko wstawiamy nowe dane
- update() - do uaktualnienia danych
- delete() - usuwa dane
- getType() - zwraca typ przechowywanych danych w formie MIME

Dostawcy treści

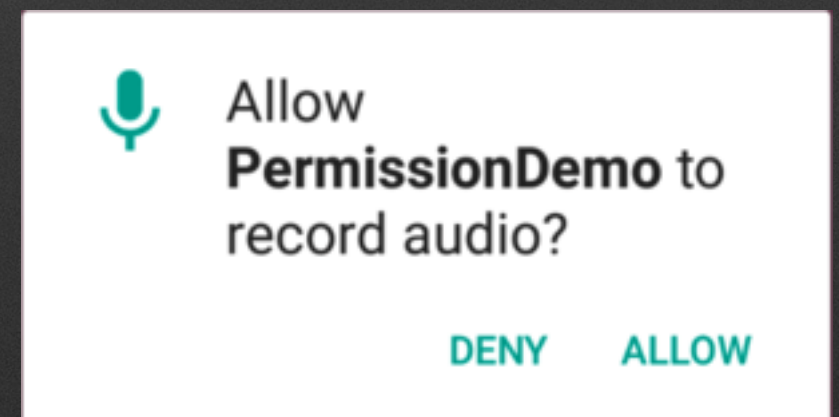
- Dostawca jest identyfikowany w systemie poprzez URI
- URI może też zawierać parametry danych do pobrania
- W komunikacji pośredniczy klasa ContentResolver, którą można uzyskać wywołując metodę `getContentResolver()`
- Dostawca musi być zdefiniowany w manifeście - znacznik `<provider>` zawierający:
 - `android:authority` - pełne URI dostawcy
 - `android:name` - nazwa klasy, często taka sama

Storage Access

- Od Androida 4.4 zaimplementowano bibliotekę Storage Access
- Umożliwia ona dostęp do kompatybilnych systemów chmurowych, np. Google Drive i Box
- Mechanizm działa na zasadzie intencji i wywoływania aktywności współpracującej z SA (tzw. Picker)

Uprawnienia aplikacji

- Do wersji 6 Androida wszystkie uprawnienia były ustawiane w manifeście aplikacji
- Od wersji 6 uprawnienia są dzielone na zwykłe (zatwierdzane w momencie instalacji) i niebezpieczne (zatwierdzane w czasie działania kodu)
- Niebezpieczne potwierdza użytkownik w oknie dialogowym



Niebezpieczne uprawnienia

GRUPA	UPRAWNIENIA
Kalendarz	odczyt, zapis
Kamera	CAMERA
Kontakty	odczyt, zapis, GET_CONTACTS
Lokacja	dokładna i przybliżona
Mikrofon	nagrywanie audio
Telefon	odczyt stanu, połączenie, rejestr połączeń, poczta głosowa, połączenia wychodzące
Czujniki	BODY_SENSORS
Telefon - wiadomości	SMS, wyślij, odbierz, czytaj, MMS, WAP_PUSH
Przechowywanie danych	odczyt i zapis z zewnętrznego źródła

Uprawnienia

- Aplikacja zawsze powinna sprawdzić, czy otrzymała uprawnienie do danej czynności - metoda `checkSelfPermission()`

```
val permission = ContextCompat.checkSelfPermission(context: this,
    Manifest.permission.RECORD_AUDIO)

if (permission != PackageManager.PERMISSION_GRANTED) {
    Log.i(TAG, msg: "Permission to record denied")
}
```

```
11:48:45.292 4559-4559/com.predki.permissionexample I/PermissionDemo: Permission to record denied
```

- Po dodaniu na starszym Androidzie uprawnień do manifestu aplikacja uprawnienia będzie miała

```
package="com.predki.permissionexample">

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

<application
```

Uprawnienia

- Od Androida 6 aplikacja musi w trakcie działania zapytać się o pozwolenie. Służą do tego metody:
 - `requestPermissions()`
 - `onRequestPermissionsResult()`
- Proces ten jest asynchroniczny

Uprawnienia

```
private fun setupPermissions() {  
    val permission = ContextCompat.checkSelfPermission(context: this,  
        Manifest.permission.RECORD_AUDIO)  
  
    if (permission != PackageManager.PERMISSION_GRANTED) {  
        Log.i(TAG, msg: "Permission to record denied, will ask")  
        ActivityCompat.requestPermissions(activity: this,  
            arrayOf(Manifest.permission.RECORD_AUDIO),  
            RECORD_REQUEST_CODE)  
    }  
}
```



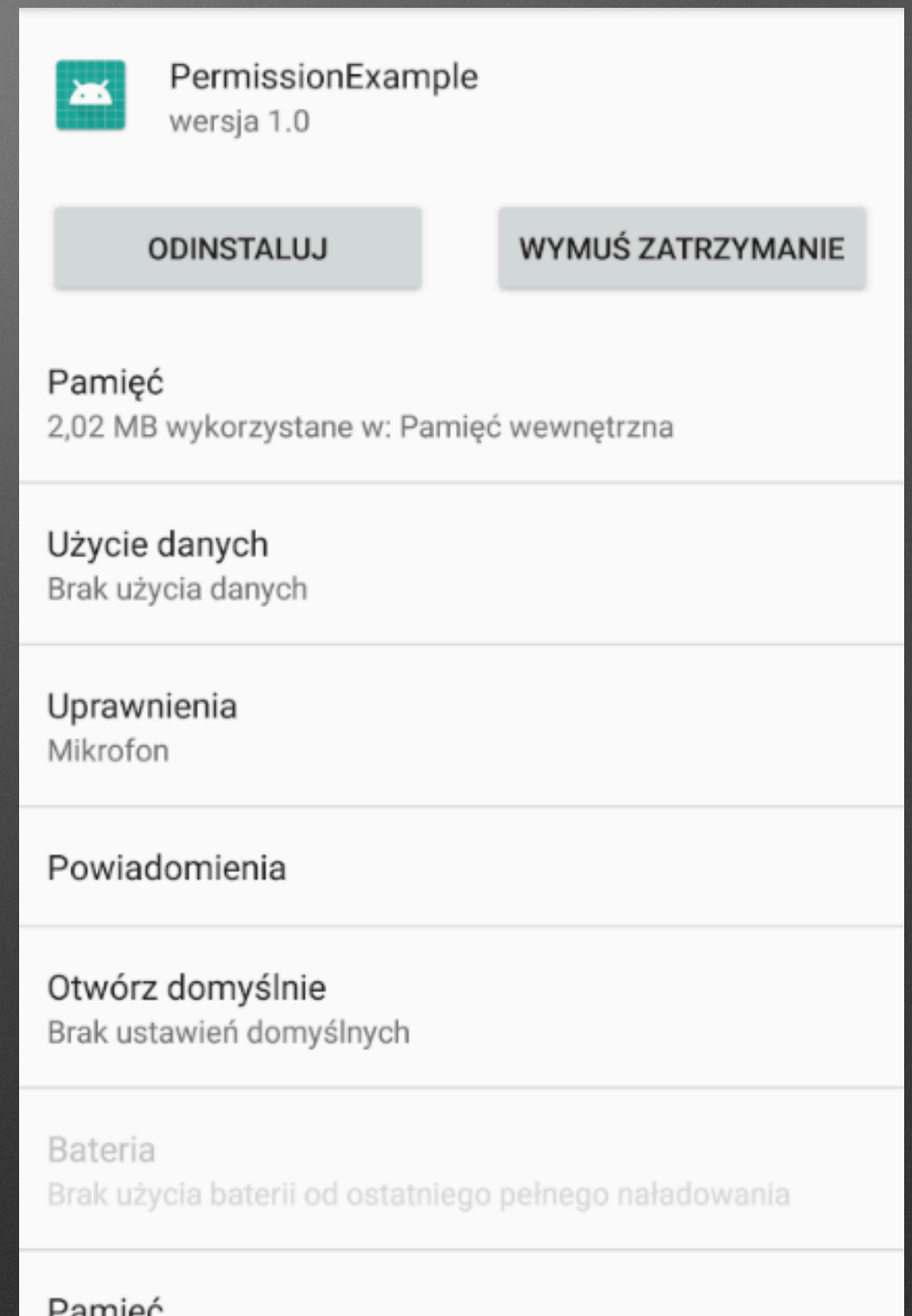
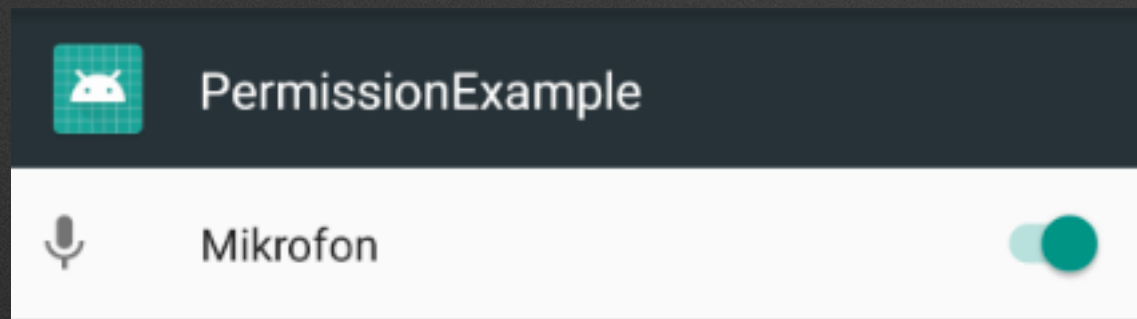
Zezwolić aplikacji
PermissionExample
na: nagrywanie
dźwięku?

ODMÓW ZEZWÓL

```
override fun onRequestPermissionsResult(requestCode: Int,  
    permissions: Array<String>, grantResults: IntArray) {  
    when (requestCode) {  
        RECORD_REQUEST_CODE -> {  
            if (grantResults.isEmpty() || grantResults[0] != PackageManager.PERMISSION_GRANTED) {  
                Log.i(TAG, msg: "Permission has been denied by user")  
            } else {  
                Log.i(TAG, msg: "Permission has been granted by user")  
            }  
        }  
    }  
}
```

Uprawnienia

- Teraz aplikacja ma już uprawnienia do ponownej instalacji lub odebrania uprawnień przez użytkownika



Uprawnienia

Permission required

Permission to access the microphone is required for this app to record audio.

OK

- Aplikacja może uzasadnić swoje żądanie uprawnień korzystając z metody `shouldShowRequestPermissionRationale()`

```
private fun setupPermissions() {  
    val permission = ContextCompat.checkSelfPermission(context: this,  
        Manifest.permission.RECORD_AUDIO)  
  
    if (permission != PackageManager.PERMISSION_GRANTED) {  
        Log.i(TAG, msg: "Permission to record denied")  
        if (ActivityCompat.shouldShowRequestPermissionRationale(activity: this,  
            Manifest.permission.RECORD_AUDIO)) {  
            val builder = AlertDialog.Builder(context: this)  
            builder.setMessage("Permission to access the microphone is required for this app to record audio.")  
                .setTitle("Permission required")  
  
            builder.setPositiveButton(text: "OK"  
            ) { dialog, id ->  
                Log.i(TAG, msg: "Clicked")  
                makeRequest()  
            }  
  
            val dialog = builder.create()  
            dialog.show()  
        } else {  
            makeRequest()  
        }  
    }  
}
```

Dostęp do aparatu

- Dostęp do aparatu możemy otrzymać wywołując odpowiednią intencję

```
fun takePicture(v:View) {  
    val intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)  
    startActivityForResult(intent, requestCode: 100)  
}
```

- Aplikacja musi posiadać uprawnienia

```
<uses-permission-sdk-23 android:name="android.permission.CAMERA" />  
<uses-permission-sdk-23 android:name="android.permission.WRITE_EXTERNAL_STORAGE" />  
<uses-permission-sdk-23 android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

- Zdjęcie dostaniemy w powrocie z aktywności

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
    if (resultCode== Activity.RESULT_OK) {  
        if (requestCode==100) {  
            if (data?.data != null) {  
            } else {  
                val image = data?.extras?.get("data") as? Bitmap  
                imageView?.setImageBitmap(image)  
            }  
        }  
    }  
}
```

Odtwarzanie mediów

- Klasa MediaPlayer - prostsza z metodami:
 - create() - inicjalizuje w oparciu o URI
 - setDataSource() - ustawia źródło
 - prepare() - przygotowuje odtwarzanie
 - start(), pause(), stop(), resume() - sterują odtwarzaniem
 - setVolume() - reguluje głośność
 - reset() - zeruje odtwarzacz
 - release() - zwalnia zasoby

Odtwarzanie mediów

- Klasa `AudioTrack` - bardziej złożona, z możliwością buforowania danych
- Dostęp do mediów wymaga uprawnień

Nagrywanie mediów

- Klasa MediaRecorder z metodami:
 - `setAudioSource()`
 - `SetVideoSource()`
 - `setOutputFormat()`
 - `setAudioEncoder()`
 - `setOutputFile()`
 - `prepare()`
 - `start()`, `stop()`
 - `reset()`, `release()`