







## Wykład Przechowywanie danych

### Programowanie Urządzeń Mobilnych

Dr inż. Damian Raczyński

## Opcje przechowywania danych

Możliwe sposoby przechowywania danych:

- Preferencje (prymitywne dane typu klucz-wartość), 
- Wewnętrzna pamięć urządzenia, 
- Zewnętrzna pamięć, 
- Dane tymczasowe, 
- Bazy danych SQLite, 
- Przechowywanie danych na serwerze. 


2


## Opcje przechowywania danych


Android tworzy prywatny katalog dla każdej aplikacji o ścieżce:


/data/data/<NAZWA.PAKIETU>/...

w katalogu znajdują się podfoldery:

database - bazy danych SQLite, 

shared\_prefs - preferencje, 


cache - dane tymczasowe 

lib - biblioteki 

files - pliki aplikacji. 

3

## Preferencje

Preferencje umożliwiają przechowywanie danych na poziomie **aktywności** (wszystkich aktywności danej aplikacji) 

Preferencje nie mogą być używane przez kod znajdujący się w innych pakietach.

Preferencje przechowywane są w postaci par nazwa-wartość, gdzie wartości mogą być typu:

- logicznego,
- zmiennoprzecinkowego,
- całkowitego,
- całkowitego typu long,
- łańcuchu znaków.



4

## Preferencje

W celu obsługi preferencji należy:

1. Pobrać obiekt `SharedPreferences`,
2. Stworzyć obiekt `SharedPreferences.Editor` (pozwoli modyfikować zawartość właściwości)
3. Wykonać modyfikację,
4. Zapisać zmiany

Aktywności mogą posiadać własne preferencje (prywatne) – przeznaczone wyłącznie dla danej aktywności. Każda aktywność może posiadać tylko jedną grupę prywatnych preferencji.

5

## Preferencje

Podstawowe metody obiektu `SharedPreferences`:

`contains` – określa, czy istnieje preferencja o podanej nazwie,

`edit` – pobiera obiekt edytora, który można użyć do modyfikacji wartości,

`getAll` – pobiera mapę (nazwa – wartość)

`getBoolean`, `getFloat`, `getInt`, `getLong`, `getString` – zwracają wartość konkretnej preferencji jako wartość określona w nazwie metody.

6

## Preferencje

Podstawowe metody obiektu edytora `SharedPreferences.Editor`:

`clear` – usuwa wszystkie preferencje,

`remove` – usuwa preferencję o podanej nazwie,

`commit` – zatwierdza wszystkie zmiany.

`putBoolean`, `putFloat`, `putInt`, `putLong`, `putString` – określa wartość preferencji o podanej nazwie.

7

## Preferencje

Funkcja:

`SharedPreferences getPreference(int mode)` zwraca obiekt dostępu do preferencji prywatnych dla bieżącej aktywności.

Wartości parametru `mode` `MODE_PRIVATE` określa, że plik może być wyłącznie dostępny przez wywołującą polecenie aplikację (do aplikacji przydzielony jest id użytkownika).

8

```

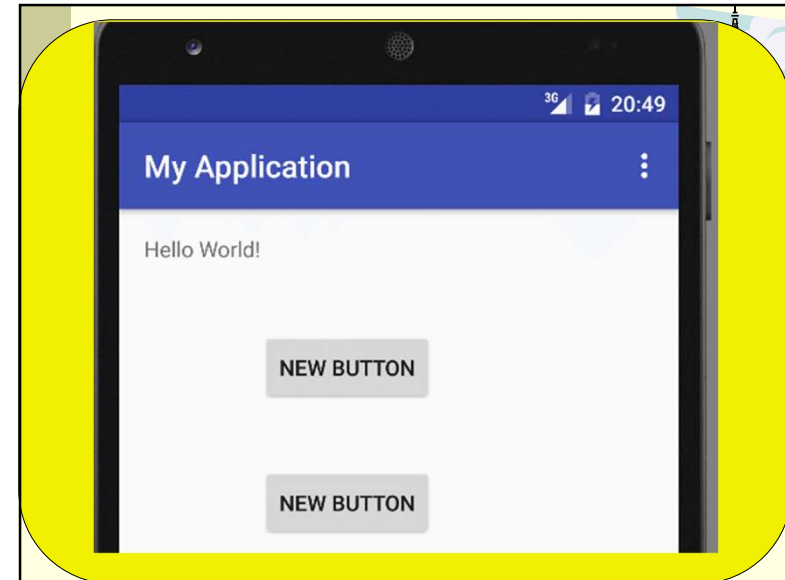
Button b1 = (Button) findViewById(R.id.button);
b1.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View arg0) {
        SharedPreferences x = getPreferences(MODE_PRIVATE);
        SharedPreferences.Editor y = x.edit();
        y.putString("preferencjal", "Przykładowa preferencja");
        y.commit();
    }
});

Button b2=(Button) findViewById(R.id.button2);
b2.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View arg0) {
        SharedPreferences x = getPreferences(MODE_PRIVATE);
        String wartosc = x.getString("preferencjal",null);
        TextView t = (TextView) findViewById(R.id.textView);
        t.setText(wartosc);
    }
});

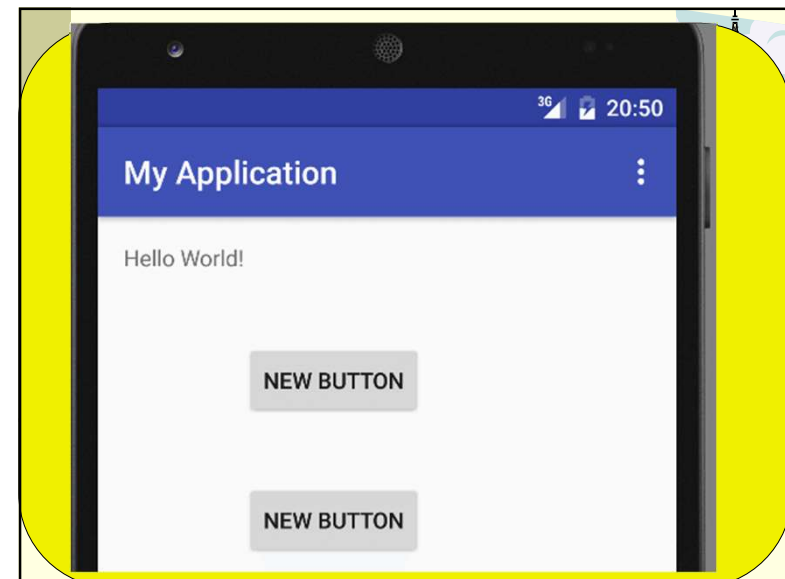
```



## Preferencje

Co się stanie po uruchomieniu aplikacji na nowo?

11



## Preferencje

Preferencje aplikacji są przechowywane w plikach XML

```
/data/data/<nazwa_pakietu>/shared_preferences/<nazwa_pliku_preferencji>.xml
```

W przypadku preferencji prywatnych nazwa pliku odpowiada nazwie aktywności

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<map>
  <string name="text_pref">Wartosc tekstowa</string>
  <int name="int_pref" value="-3234554"/>
  <float name="float_pref" value="33.545"/>
</map>
```

13

## Operacje na plikach

Dane aplikacji android przechowywane są w systemie plików:

```
/data/data/<nazwa_pakietu>/
```

W katalogu tworzonych jest szereg podkatalogów do przechowywania baz danych, preferencji i innych plików.

Do wykonywania operacji na plikach wykorzystywany jest obiekt Context

14

## Operacje na plikach

Podstawowe metody obiektu Context do operacji na plikach (pliki znajdują się w podkatalogu files):

openFileInput – Otwiera plik do odczytu,

openFileOutput – otwiera plik do zapisu,

deleteFile – usuwa plik,

fileList – zwraca listę plików,

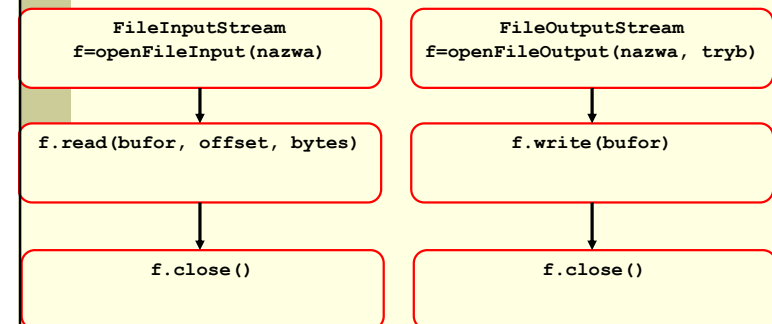
getFilesDir – zwraca obiekt podkatalogu,

getDir – pobiera lub tworzy podkatalog o podanej nazwie

15

## Operacje na plikach

Ogólny schemat postępowania przy zapisie odczycie pliku w pamięci wewnętrznej:



## Operacje na plikach



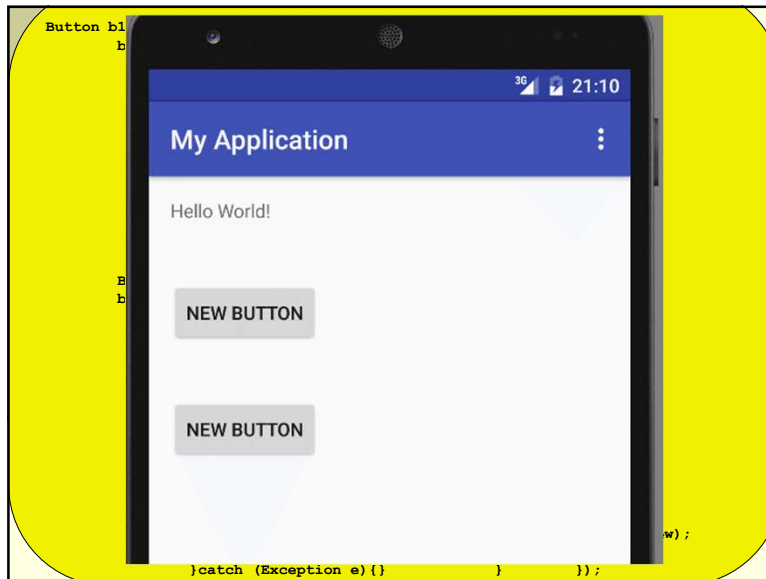
Tryby dostępu do plików dla `openFileOutput`:

`MODE_PRIVATE` – dostęp do pliku tylko dla aplikacji wywołującej polecenie

`MODE_APPEND` – otwiera plik do zapisu, jeżeli plik istnieje to dane dopisywane są na koniec pliku (zamiast zamazywać zawartość bieżącą).

17

```
Button b1 = (Button) findViewById(R.id.button);
b1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        try{
            FileOutputStream plk;
            String tekst = "Programowanie urządzeń mobilnych";
            plk = openFileOutput("pum.txt", MODE_PRIVATE);
            plk.write(tekst.getBytes());
            plk.close();
            plk=openFileOutput("pum.txt", MODE_APPEND);
            plk.write(new String("JEE także").getBytes());
            plk.close();
        } catch (Exception e){}
    }
});
Button b2=(Button) findViewById(R.id.button2);
b2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        try{
            FileInputStream plk = openFileInput("pum.txt");
            StringBuffer buf = new StringBuffer();
            InputStreamReader x = new InputStreamReader(plk);
            BufferedReader d = new BufferedReader(x);
            String linia = null;
            while ((linia=d.readLine())!=null){
                buf.append(linia+"\n");
            }
            d.close(); x.close(); plk.close();
            TextView t = (TextView) findViewById(R.id.textView);
            t.setText(buf.toString());
        } catch (Exception e){}
    }
});
```



## Operacje na plikach



W przypadku, gdy aplikacja musi realizować inne operacje na plikach, to należy wykorzystać klasę `java.io.File`

przykładowo:

```
import java.io.File;
...
File x = getFilesDir(); → data/data/<PAKIET>/files
String[] lista = x.list();
```

(lista wszystkich elementów w katalogu files)

20

java.io.File	
Metoda	Opis
canExecute, canRead, canWrite	Sprawdzają, czy aplikacja może uruchomić/odczytać/zapisać określony plik
createNewFile	Tworzy plik, jeżeli go nie było
delete	usuwa plik/katalog
exists	sprawdza czy plik/katalog istnieje
getAbsolutePath	zwraca ścieżkę bezwzględną pliku
getName	zwraca nazwę pliku/katalogu
isDirectory, isFile	sprawdza czy element jest plikiem/katalogiem
length	zwraca rozmiar pliku
String [] list	Zwraca listę plików i katalogów w katalogu
File[] listFile	Zwraca tablicę obiektów File w folderze
mkdir	tworzy katalog
mkdirs	tworzy katalog wraz z katalogami podanymi w ścieżce (jeżeli nie istniały)
File(...)	File(String pathname), File(String parent, String child - konstruktory.

## Operacje na plikach

data/data/<PAKIET>/files

W przypadku, gdy aplikacja dysponuje uprawnieniami do tworzenia plików w innym miejscu, niż katalog files:

```
import java.io.File;
import java.io.FileOutputStream;
...
File kat = getFilesDir();
String plk_nam="plik.dat";
String text="PUM";
File nowy = new File(kat, plk_nam);
nowy.createNewFile();
FileOutputStream x = new
FileOutputStream(nowy.getAbsolutePath());
x.write(text.getBytes());
```

## Operacje na plikach

```
File kat=getFilesDir();
String plk_nam="plik.dat";
String text="PUM";
File nowy = new File(kat, plk_nam);
try {
    nowy.createNewFile();
    FileOutputStream x = new FileOutputStream(nowy.getAbsolutePath());
    x.write(text.getBytes());
    String katalog=kat.getAbsolutePath();
} catch (IOException e) {
    e.printStackTrace();
}
```

23

## Dostęp do pamięci zewnętrznej

W celu odczytu/zapisu danych na pamięci zewnętrznej, aplikacja musi w pliku manifestu zamieścić:

### ODCZYT I ZAPIS:

```
<manifest ...>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>
```



### TYLKO ODCZYT:

```
<manifest ...>
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
</manifest>
```



Od Android 4.4 uprawnienia te nie są wymagane jeżeli odczyt/zapis dotyczy plików prywatnych aplikacji.

24

## Dostęp do pamięci zewnętrznej

Sprawdzanie dostępności pamięci zewnętrznej:

Obiekt `Environment` umożliwia dostęp do zmiennych środowiskowych. Metoda `getExternalStorageState()` zwraca stan głównej pamięci zewnętrznej.

```
TextView t= (TextView) findViewById(R.id.textView);
String state= Environment.getExternalStorageState();
if(Environment.MEDIA_MOUNTED.equals(state)){
    t.setText("External memory mounted");
}
```

External memory mounted

25

## Dostęp do pamięci zewnętrznej

Zapisywanie plików współdzielonych z innymi aplikacjami

Aby uzyskać obiekt `File` reprezentujący publiczny katalog należy wywołać metodę `getExternalStoragePublicDirectory()`, przekazując do metody typ katalogu (np. `DIRECTORY_MUSIC`, `DIRECTORY_PICTURES`, `DIRECTORY_RINGTONES`, ...).



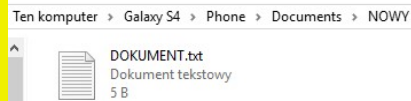
Metoda `getExternalStoragePublicDirectory` może zwrócić ścieżkę do folderu publicznego w pamięci urządzenia (nie necessarily na karcie SD).



26

## Dostęp do pamięci zewnętrznej

```
File file= new File(Environment.getExternalStoragePublicDirectory(
Environment.DIRECTORY_DOCUMENTS), "NOWY");
if(!file.mkdirs()){
    Toast.makeText(this, "Katalog istnieje", Toast.LENGTH_LONG).show();
}
File nowy = new File(file, "DOKUMENT.txt");
if(!nowy.exists()) {
    try {
        nowy.createNewFile();
        FileOutputStream x = new
FileOutputStream(nowy.getAbsolutePath());
        x.write("TEKST".getBytes());
        x.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```



## Dostęp do pamięci zewnętrznej

Zapisywanie plików prywatnych (nieдоступnych dla innych aplikacji)



metoda `getExternalFilesDir` zwraca prywatny katalog na pamięci zewnętrznej.



Metoda pobiera argument określający typ katalogu (np. `DIRECTORY_MOVIES`).



Przekazanie wartości `null` spowoduje wynik w postaci prywatnego katalogu głównego dla aplikacji.



Gdy użytkownik odinstalowuje aplikację, katalog z całą zawartością jest usuwany.



Skaner systemowy nie ma dostępu do tego położenia (pliki nie będą dostępne przez `MediaStore`) - pliki użytkownika powinny zostać zapisane w katalogu publicznym.



28

## Dostęp do pamięci zewnętrznej

```
TextView t= (TextView) findViewById(R.id.textView);
File file= new File(getExternalFilesDir(null), "NOWY");
t.setText(file.getAbsolutePath());
if(!file.mkdirs()){
    Toast.makeText(this, "Katalog istnieje",
        Toast.LENGTH_LONG).show();
}
File nowy = new File(file, "DOKUMENT.txt");
if(!nowy.exists()) {
    try {
        nowy.createNewFile();
        FileOutputStream x = new
FileOutputStream(nowy.getAbsolutePath());
        x.write("TEKST".getBytes());
        x.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
```

/storage/emulated/0/Android/data/pl.nysa.pwsz.animatory/files/NOWY

## Dostęp do pamięci zewnętrznej

Metoda `getExternalFilesDir()` (minimum API 19 - Android - Android 4.4) zwraca tablicę obiektów klasy `File`, która zawiera możliwe lokalizacje w pamięci zewnętrznej. Alternatywnie `ContextCompat.getExternalFilesDirs()` (Android 4.3 i niższe):

```
TextView t= (TextView) findViewById(R.id.textView);
File files []=null;
if(Build.VERSION.SDK_INT>Build.VERSION_CODES.KITKAT)
    files= getExternalFilesDirs(null);
else
    files= ContextCompat.getExternalFilesDirs(this, null);
t.setText("");
for(File x : files){
    t.setText(t.getText()+"\n"+x.getAbsolutePath());
}
}
```

/storage/emulated/0/Android/data/pl.nysa.pwsz.animatory/files  
/storage/3238-3064/Android/data/pl.nysa.pwsz.animatory/files

30

## Pliki cache

W przypadku potrzeby przechowywania pewnych danych tymczasowych należy wykorzystać metodę `getCacheDir()` w celu pobrania katalogu zawierającego dane tymczasowe.



Gdy urządzeniu zaczyna brakować pamięci wewnętrznej, Android może usunąć pliki z katalogu tymczasowego.



Powinno się zarządzać samemu plikami tymczasowymi (nie należy polegać na działaniach systemu).



Gdy użytkownik usuwa aplikację, pliki tymczasowe są usuwane.



W odniesieniu do pamięci zewnętrznej:

`getExternalCacheDir()` / `ContextCompat.getExternalCacheDirs()`



```
TextView t= (TextView) findViewById(R.id.textView);
File file=null;
if(Build.VERSION.SDK_INT>Build.VERSION_CODES.KITKAT)
    file= getCacheDir();
else
    file= ContextCompat.getExternalCacheDirs(this)[0];
t.setText(file.getAbsolutePath());
File cache = new File(file, "cache");
if(!cache.exists()){
    try {
        cache.createNewFile();
        FileOutputStream os= new FileOutputStream(cache.getAbsolutePath());
        os.write("cache".getBytes());
        os.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
//.....
cache.delete();
```

/data/user/0/pl.nysa.pwsz.animatory/cache



## SQLite



Bazy danych SQLite bazują na wykorzystaniu plików (bez konieczności uruchamiania oddzielnego procesu serwera) – nadają się do zastosowań dla urządzeń mobilnych.



SQLite udostępnia transakcje ACID oraz większość standardu SQL 92.



Bazy danych zapisywane są jako pliki binarne, ich bezpieczeństwo jest oparte na zabezpieczeniach oferowanych przez system.



W Android informacje są prywatne (dostępne tylko dla konkretnej aplikacji).



W przypadku chęci udostępnienia informacji – należy aplikację zadeklarować jako dostawcę treści.



## SQLite



Tworzenie obiektu bazy danych:

Wywołanie metody `openOrCreateDatabase()` obiektu `Context`:



```
SQLiteDatabase baza;  
baza= openOrCreateDatabase("nazwa.db",  
SQLiteDatabase.CREATE_IF_NECESSARY, null);
```

Aplikacje zapisują bazy w katalogu:

`/data/data/<nazwa pakietu>/databases/<nazwa bazy>`



34

## SQLite



Po utworzeniu obiektu `SQLiteDatabase` należy go skonfigurować



- ustawienia lokalne,
- mechanizmy blokowania dla aplikacji wielowątkowych,
- numer wersji.



```
SQLiteDatabase baza;  
baza = openOrCreateDatabase("nazwa.db",  
SQLiteDatabase.CREATE_IF_NECESSARY, null);  
baza.setLocale(Locale.getDefault());  
baza.setLockingEnabled(true);  
baza.setVersion(1);
```

35

## SQLite



Tworzenie tabel oraz innych obiektów baz danych SQLite związane jest wywołaniem kodu SQLite.



```
String create="create table tabela(  
id integer primary key autoincrement,  
kolumna2 text,  
kolumna3 text);";  
baza.execSQL(create);
```

Metoda `execSQL` umożliwia wykonywanie poleceń SQL, które nie są związane z zapytaniami (np. tworzenie, modyfikowanie oraz usuwanie tabel, widoków, wyzwalaczy ...)

36

## SQLite



### Dodawanie rekordów

Do dodawania nowych elementów tabeli służy metoda insert

Wartości dodawane do tabeli są kojarzone z konkretnymi tabelami z wykorzystaniem obiektu ContentValues.



```
ContentValues x = new ContentValues();
x.put("nazwa_kol", "wartosc");
...
long id = db.insert("tabela", null, x);
```

37

## SQLite



### Wybrane metody obiektu ContentValues

metoda	opis
clear()	usuwa wszystkie wartości
boolean containsKey(String key)	określa istnienie klucza
Object get(String key)	zwraca wartość związaną z kluczem
typ getAsTyp(String key) gdzie typ oznacza podstawowe typy	zwraca wartość o określonym typie
Set <String> keySet()	zwraca kolekcję kluczy
void put(String key, typ wartość)	umieszcza parę klucz-wartość w danych obiektu
void remove(String key)	usuwa element z obiektu
int size()	zwraca liczbę elementów
Set<Entry<String, Object>> valueSet()	zwraca kolekcję par klucz-wartość

## SQLite



Button przycisk  
przycisk.setOnClick  
@Override  
public void onClick()  
{  
 SQLiteDatabase db = openOrCreateDatabase("db", Context.MODE\_PRIVATE, null);  
 db.execSQL("CREATE TABLE IF NOT EXISTS studenci (id INTEGER PRIMARY KEY AUTOINCREMENT, imie TEXT, nazwisko TEXT);");  
 ContentValues cv = new ContentValues();  
 cv.put("imie", "Jan");  
 cv.put("nazwisko", "Kowalski");  
 long id = db.insert("studenci", null, cv);  
}





Przy ponownym naciśnięciu przycisku kod będzie stanowił problem, gdyż tabela już będzie istniała!

## SQLite



### Aktualizowanie wierszy

Metoda update pobiera następujące parametry:

- Nazwa modyfikowanej tabeli, 
- Obiekt ContentValues zawierające nowe wartości pól, 
- Opcjonalna klauzula where (znaki ? Reprezentują argumenty), 
- tabela argumentów dla klauzuli where 

W przypadku braku klauzuli where (null), zmiany będą dotyczyły wszystkich rekordów.

40

## SQLite






```
Button przycisk = (Button) findViewById(R.id.button);
przycisk.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        SQLiteDatabase db;
        db=openOrCreateDatabase("db2.db",
        SQLiteDatabase.CREATE_IF_NECESSARY, null);
        ContentValues cv = new ContentValues();
        cv.put("imie", "Jan");
        cv.put("nazwisko", "Malinowski");
        db.update("studenci", cv, "nazwisko=?", new
        String[]{"Kowalski"});
    }
});
```

## SQLite



### Usuwanie wierszy

Parametry metody delete:

- Nazwa modyfikowanej tabeli, 
- Opcjonalna klauzula where (znaki ? Reprezentują argumenty), 
- tabela argumentów dla klauzuli where 

W przypadku braku klauzuli where (null), wszystkie rekordy danej tabeli zostaną usunięte.

42

## SQLite




```
Button przycisk = (Button) findViewById(R.id.button);
przycisk.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        SQLiteDatabase db;
        db=openOrCreateDatabase("db2.db",
        SQLiteDatabase.CREATE_IF_NECESSARY, null);
        db.delete("studenci", "imie=?", new String[]{"adam"});
    }
});
```

43

## SQLite



### Zapytania SQL

Do odczytywania wyników zapytania wykorzystywany jest obiekt Cursor (pozwala na swobodny dostęp do wyników zapytania zwróconych przez bazę). 

Dostęp do metod obiektu **nie** jest synchronizowany (w przypadku dostępu do obiektu przez wiele wątków należy zaimplementować synchronizację samemu).

Każdy wiersz odpowiada jednemu zwróconemu rekordowi.

```
Cursor x = db.query("studenci", null, null, null,
null, null, null); // select * from studenci
```

```
// operacje
x.close();
```

44

SQLite Wybrane metody Cursor	
metoda	opis
<code>close()</code>	zamyka kursor, zwalnia zasoby
<code>int getColumnCount()</code>	zwraca liczbę kolumn
<code>int getColumnIndex(String name)</code>	zwraca indeks kolumny o konkretnej nazwie
<code>String getColumnName(int index)</code>	zwraca nazwę kolumny
<code>String [] getColumnNames()</code>	zwraca tablicę nazw kolumn
<code>int getCount()</code>	liczba wierszy
<b>TYP</b> <code>getTYP(int index)</code>	pobiera wartość z kolumny index
<code>int getPositon()</code>	zwraca bieżącą pozycję w zbiorze wierszy
<code>boolean isAfterLast()</code>	czy kursor wskazuje na pozycję za ostatnim elementem zbioru
<code>boolean isBeforeFirst()</code>	czy kursor wskazuje pozycję przed pierwszym elementem
<code>boolean isClosed()</code>	czy kursor został zamknięty

Wybrane metody Cursor SQLite	
metoda	opis
<code>boolean isFirst()</code> , <code>boolean isLast()</code>	czy pierwszy lub ostatni
<code>boolean isNull(int index)</code>	czy element w kolumnie index jest równy null
<code>boolean move(int offset)</code>	przenosi kursor o określoną liczbę pozycji do przodu lub tyłu (względem bieżącej pozycji)
<code>boolean moveToFirst()</code> , <code>boolean moveToLast()</code>	przenosi kursor do pierwszego/ostatniego wiersza
<code>boolean moveToNext()</code>	przenosi kursor do następnego wiersza
<code>boolean moveToPositon(int position)</code>	przenosi kursor do pozycji bezwzględnej (liczona od początku danych)
<code>boolean moveToPrevious()</code>	przenosi kursor do poprzedniego wiersza

```

Button przycisk = (Button) findViewById(R.id.button);
przycisk.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        SQLiteDatabase db;
        db=openOrCreateDatabase("db2.db",
        SQLiteDatabase.CREATE_IF_NECESSARY, null);
        Cursor c = db.query("studenci", null, null, null, null,
        null, null);
        c.moveToFirst();
        String wiersz="";
        while(c.isAfterLast()==false)
        {
            for (int i=0; i<c.getColumnCount(); i++)
                wiersz=wiersz+" "+c.getString(i);
            c.moveToNext();
            wiersz+="\n";
        }
        TextView t = (TextView) findViewById(R.id.textView);
        t.setText(wiersz);
    }
});

```

