







## 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

```

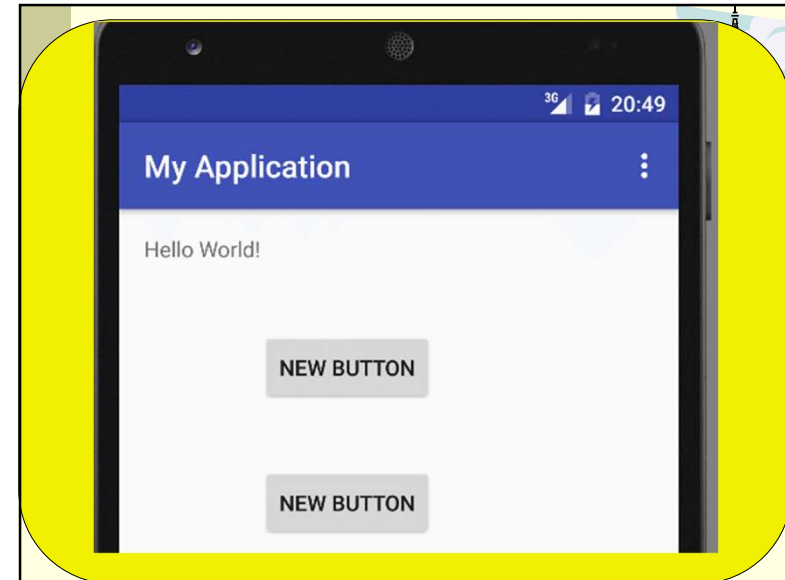
Button b1 = (Button) findViewById(R.id.button);
b1.setOnClickListener(new View.OnClickListener() {

    @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);
    }
});

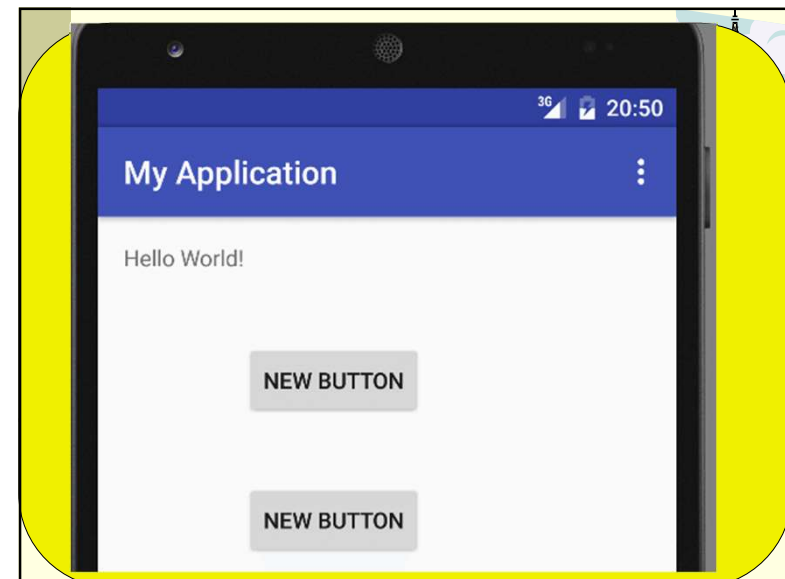
```



## 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

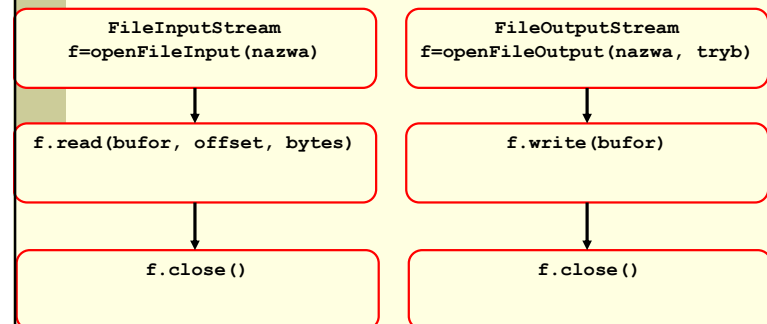


15

## Operacje na plikach



Ogólny schemat postępowania przy zapisie odczycie pliku w pamięci wewnętrznej:



## Operacje na plikach



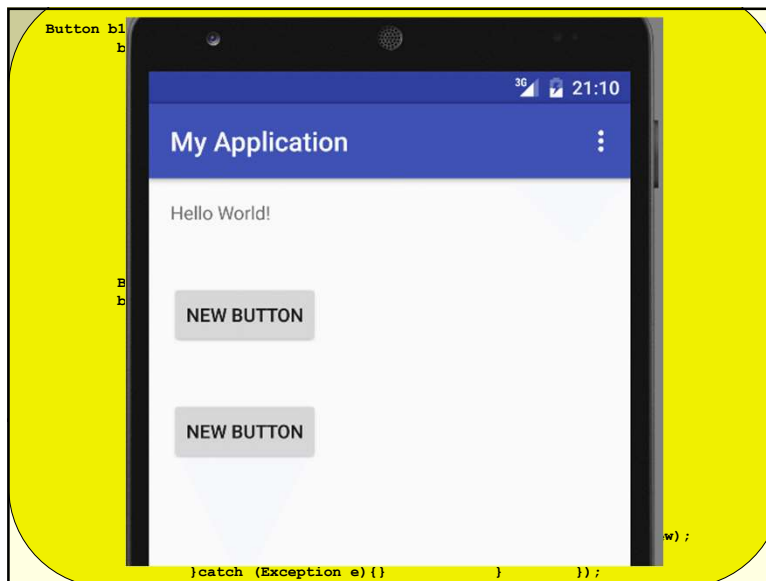
Tryby dostępu do plików dla `openFileOutput`:

`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
makedirs	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());
```

22

## 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`, ...).



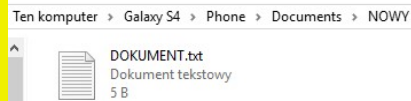
Metoda `getExternalStoragePublicDirectory` może zwrócić ścieżkę do folderu publicznego w pamięci urządzenia (nie necessarily na karcie SD).



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

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

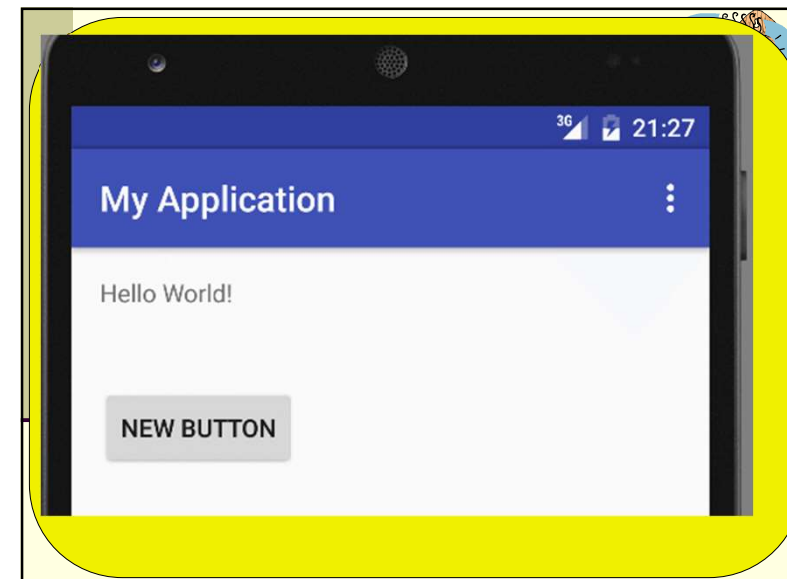
SQLite Wybrane metody Cursor	
metoda	opis
<code>close()</code>	zamyka kursor, zwalnia zasoby
<code>int getColumnCount()</code>	zwraca liczbę kolumn
<code>int getColumnIndex(String name)</code>	zwraca indeks kolumny o konkretnej nazwie
<code>String getColumnName(int index)</code>	zwraca nazwę kolumny
<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);
    }
});

```



## Parametry metody query:



String	• Nazwa tabeli
String []	• lista kolumn
String	• klauzula where (zapis ze znakami ? definiujące warunki)
String[]	• wartości dla zdefiniowanych wcześniej warunków
String	• parametry dla klauzuli group by
String	• parametry dla klauzuli having
String	• parametry dla klauzuli order by
String	• parametry dla klauzuli limit

## SQLite



### Przykład:

```
Cursor c = db.query("studenci", null, "imie=?", new
String[]{„Krzysztof"}, null, null, null);
```

### odpowiednik:

```
select * from studenci where imie=„Krzysztof"
```

```
String kolumny[] = {"imie", "id"};
String sortowa="imie ASC";
Cursor x = db.query("studenci", kolumny, null, null,
null, null, sortowa);
```

### odpowiednik:


```
Select imie, id from studenci order by imie ASC;
```

50

```
TextView t= (TextView) findViewById(R.id.textView);
SQLiteDatabase db;
db=openOrCreateDatabase("db.db", SQLiteDatabase.CREATE_IF_NECESSARY, null);
db.execSQL("create table kolokwium(id integer primary key autoincrement, imie
text, nazwisko text, ocena real);");
ContentValues cv= new ContentValues();
cv.put("imie", "Jan"); cv.put("nazwisko", "Nowak"); cv.put("ocena", 5.0);
long id=db.insert("kolokwium", null, cv);
cv.put("imie", "Adam"); cv.put("nazwisko", "Kowalski"); cv.put("ocena", 4.0);
id=db.insert("kolokwium", null, cv);
cv.put("imie", "Marek"); cv.put("nazwisko", "Nowak"); cv.put("ocena", 3.0);
id=db.insert("kolokwium", null, cv);
cv.put("imie", "Anna"); cv.put("nazwisko", "Malinowska"); cv.put("ocena",
2.0);
id=db.insert("kolokwium", null, cv);
cv.put("imie", "Iwona"); cv.put("nazwisko", "Malinowska"); cv.put("ocena",
5.0);
id=db.insert("kolokwium", null, cv);
Cursor c = db.query("kolokwium", 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";
    }
    t.setText(wiersz);
}
```

```
1 Jan Nowak 5
2 Adam Kowalski 4
3 Marek Nowak 3
4 Anna Malinowska 2
5 Iwona Malinowska 5
```

```
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";
    }
    t.setText(wiersz);
}
```




```

1 Jan Nowak 5
2 Adam Kowalski 4
3 Marek Nowak 3
4 Anna Malinowska 2
5 Iwona Malinowska 5

TextView t= (TextView) findViewById(R.id.textView);
SQLiteDatabase db;
db=openOrCreateDatabase("db.db", SQLiteDatabase.CREATE_IF_NECESSARY, null);
Cursor c = db.query("kolokwium",new String[]{"imie", "nazwisko"}, 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";
}
t.setText(wiersz);

```

Jan Nowak  
Adam Kowalski  
Marek Nowak  
Anna Malinowska  
Iwona Malinowska




```

1 Jan Nowak 5
2 Adam Kowalski 4
3 Marek Nowak 3
4 Anna Malinowska 2
5 Iwona Malinowska 5

TextView t= (TextView) findViewById(R.id.textView);
SQLiteDatabase db;
db=openOrCreateDatabase("db.db", SQLiteDatabase.CREATE_IF_NECESSARY, null);
Cursor c = db.query("kolokwium",null, "imie=? or nazwisko=?", new String[]
{"Jan", "Malinowska"}, 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";
}
t.setText(wiersz);

```

1 Jan Nowak 5  
4 Anna Malinowska 2  
5 Iwona Malinowska 5




```

1 Jan Nowak 5
2 Adam Kowalski 4
3 Marek Nowak 3
4 Anna Malinowska 2
5 Iwona Malinowska 5

TextView t= (TextView) findViewById(R.id.textView);
SQLiteDatabase db;
db=openOrCreateDatabase("db.db", SQLiteDatabase.CREATE_IF_NECESSARY, null);
Cursor c = db.query("kolokwium",new String[]{"nazwisko"}, null, null,
"nazwisko", "COUNT(nazwisko)>=2", 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";
}
t.setText(wiersz);

```

Malinowska  
Nowak



```

1 Jan Nowak 5
2 Adam Kowalski 4
3 Marek Nowak 3
4 Anna Malinowska 2
5 Iwona Malinowska 5

TextView t= (TextView) findViewById(R.id.textView);
SQLiteDatabase db;
db=openOrCreateDatabase("db.db", SQLiteDatabase.CREATE_IF_NECESSARY, null);
Cursor c = db.query("kolokwium",null, null, null, null, null, "ocena ASC");
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";
}
t.setText(wiersz);

```

4 Anna Malinowska 2  
3 Marek Nowak 3  
2 Adam Kowalski 4  
1 Jan Nowak 5  
5 Iwona Malinowska 5

## SQLite



Dla bardziej skomplikowanych zapytań (np. operujących na kilku tabelach) należy wykorzystać:

SQLiteQueryBuilder – programowe tworzenie skomplikowanych zapytań, np:

metoda	opis
void appendWhere(where)	dodaje cały warunek klauzuli where
void setTables(String t)	lista tabel zapytania
Cursor query(db, kolumny, where_?, where_parm, groupBy, having, sort, limit)	parametry identyczne do metody query obiektu SQLiteDatabase - metoda zwraca Cursor
String buildQuery(kolumny, where_?, where_parm, groupBy, having, sort, limit)	parametry podobne do metody query obiektu SQLiteDatabase - metoda zwraca zapytanie w postaci String

## SQLite



Przykład wykorzystania:

```
SQLiteQueryBuilder x = new SQLiteQueryBuilder();
x.setTables("studenci, kierunki");
x.appendWhere("studenci.kierunek=kierunki.id");
String kolumny [] ={"studenci.imie",
"studenci.nazwisko", "kierunki.nazwa",
"kierunki.budynek"};
String sort = "imie ASC";
Cursor c=x.query(db, kolumny, null, null, null,
null, sort);
```

58

```
TextView t= (TextView) findViewById(R.id.textView);
SQLiteDatabase db;
db=openOrCreateDatabase("db.db", SQLiteDatabase.CREATE_IF_NECESSARY,
null);
db.execSQL("create table info(id integer primary key autoincrement,
poprawne integer, bledne integer);");
ContentValues cv= new ContentValues();
cv.put("poprawne",30); cv.put("bledne", 0);
long id=db.insert("info", null, cv);
cv.put("poprawne",20); cv.put("bledne", 10);
id=db.insert("info", null, cv);
cv.put("poprawne",10); cv.put("bledne", 20);
id=db.insert("info", null, cv);
cv.put("poprawne",0); cv.put("bledne", 30);
id=db.insert("info", null, cv);
cv.put("poprawne",30); cv.put("bledne", 0);
id=db.insert("info", null, cv);
Cursor c = db.query("info", 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";
}
t.setText(wiersz);
```

1	30	0
2	20	10
3	10	20
4	0	30
5	30	0

1	Jan Nowak	5
2	Adam Kowalski	4
3	Marek Nowak	3
4	Anna Malinowska	2
5	Iwona Malinowska	5


1	30	0
2	20	10
3	10	20
4	0	30
5	30	0



```
TextView t= (TextView) findViewById(R.id.textView);
SQLiteDatabase db;
db=openOrCreateDatabase("db.db", SQLiteDatabase.CREATE_IF_NECESSARY, null);
SQLiteQueryBuilder x= new SQLiteQueryBuilder();
x.setTables("kolokwium,info");
x.appendWhere("kolokwium.id=info.id");
Cursor c=x.query(db,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";
}
t.setText(wiersz);
```

1	Jan Nowak	5	1	30	0
2	Adam Kowalski	4	2	20	10
3	Marek Nowak	3	3	10	20
4	Anna Malinowska	2	4	0	30
5	Iwona Malinowska	5	5	30	0

1	Jan Nowak	5	1	30	0
2	Adam Kowalski	4	2	20	10
3	Marek Nowak	3	3	10	20
4	Anna Malinowska	2	4	0	30
5	Iwona Malinowska	5	5	30	0




```

TextView t= (TextView) findViewById(R.id.textView);
SQLiteDatabase db;
db=openOrCreateDatabase("db.db", SQLiteDatabase.CREATE_IF_NECESSARY, null);
SQLiteQueryBuilder x= new SQLiteQueryBuilder();
x.setTables("kolokwium,info");
x.appendWhere("kolokwium.id=info.id");
Cursor c=x.query(db,new String[]{"nazwisko", "poprawne"}, 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";
}
t.setText(wiersz);
  
```

Nowak 30  
 Kowalski 20  
 Nowak 10  
 Malinowska 0  
 Malinowska 30

## SQLite



W przypadku chęci przekazania zapytania w postaci zwykłego kodu SQL możliwe jest wykorzystanie metody `rawQuery`.


np.

```

String zapyt = „Select imie, nazwisko form
studenci where id=?”;
Cursor c = db.rawQuery(zapyt, new String[]
{„15”});
  
```

62

## SQLite



**Usuwanie tabel**

**Możliwe użycie metody `execSQL`:**

```
db.execSQL(„drop table studenci”);
```

**Zamykanie bazy danych**




**close:**

```
db.close();
```

**Usuwanie baz danych:**


**metoda `deleteDatabase` kontekstu:**

```
deleteDatabase(„db.db”);
```

63

## SQLite



**Transakcje**

**Metoda `beginTransaction`:**

```

db.beginTransaction();
try{
    // operacje na bazie danych
    db.setTransactionSuccessful();
} catch (Exception e){
    // niepowodzenie
} finally{
    db.endTransaction();
}
  
```

64



```

TextView t= (TextView) findViewById(R.id.textView);
SQLiteDatabase db;
db=openOrCreateDatabase("db.db", SQLiteDatabase.CREATE_IF_NECESSARY, null);
db.beginTransaction();
boolean wyjatek=true;
try{
    ContentValues cv= new ContentValues();
    cv.put("poprawne",100); cv.put("bledne", 100);
    db.insert("info", null, cv);
    if(wyjatek==true) throw new Exception("wyjatek");
    cv.put("poprawne",1000); cv.put("bledne", 1000);
    db.insert("info", null, cv);
    db.setTransactionSuccessful();
}catch(Exception e){
    Toast.makeText(this, "WYJATEK!!!", Toast.LENGTH_LONG).show();
} finally{
    db.endTransaction();
}
Cursor c = db.query("info",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";
}
t.setText(wiersz);

```

```

TextView t= (TextView) findViewById(R.id.textView);
SQLiteDatabase db;
db=openOrCreateDatabase("db.db", SQLiteDatabase.CREATE_IF_NECESSARY, null);
db.beginTransaction();
boolean wyjatek=true;
try{
    ContentValues cv= new ContentValues();
    cv.put("poprawne",100); cv.put("bledne", 100);
    db.insert("info", null, cv);
    if(wyjatek==true) throw new Exception("wyjatek");
    cv.put("poprawne",1000); cv.put("bledne", 1000);
    db.insert("info", null, cv);
    db.setTransactionSuccessful();
}catch(Exception e){
    Toast.makeText(this, "WYJATEK!!!", Toast.LENGTH_LONG).show();
} finally{
    db.endTransaction();
}
Cursor c = db.query("info",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";
}
t.setText(wiersz);

```

## Biblioteka Room

Należy dodać odpowiednie zależności:



The image shows a snippet of a Gradle file with the following content:

```

allprojects {
    repositories {
        jcenter()
        maven{url 'https://maven.google.com'}
    }
}

dependencies {
    compile 'android.arch.persistence.room:runtime:1.0.0-rc1'
    annotationProcessor 'android.arch.persistence.room:compiler:1.0.0-rc1'
}

```

Arrows point from the text "Należy dodać odpowiednie zależności:" to the Gradle Scripts section and the dependencies section.

## Biblioteka Room

Biblioteka Room dostarcza warstwę ponad bazą SQLite do "bezSQL'owego" dostępu do bazy danych.

W Room występują 3 główne komponenty:



1. Database - zawiera uchwyt bazy danych. Klasa poprzedzona adnotacją @Database powinna spełniać warunki:

- być klasą abstrakcyjną dziedziczącą po RoomDatabase,

- Zawierać listę encji związanych z bazą danych **przez adnotacje**,

- Zawierać abstrakcyjną metodą bezparametrową, która zwraca klasę poprzedzoną adnotacją @Dao (typ zwracanej klasy determinuje interfejs do bazy).

## Biblioteka Room

Instancję klasy Database można uzyskać wywołując:  
`Room.databaseBuilder()` lub  
`Room.inMemoryDatabaseBuilder()`.



2. Entity - reprezentuje tabelę w bazie

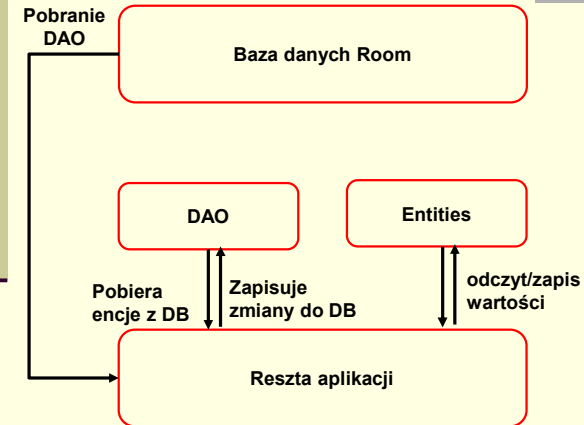


3. DAO - zawiera metody wykorzystywane do dostępu do bazy danych



69

## Biblioteka Room



70

## Biblioteka Room

Encję stanowi zbiór powiązanych ze sobą pól (tabela).



Dla każdej encji, w powiązanym obiekcie Database tworzona jest tabela.



Dla każdego pola encji tworzona jest kolumna. Jeżeli klasa ma pola, które nie powinny zostać zapisywane do tabeli należy je poprzedzić adnotacją `@Ignore`.



```

@Entity
class Student{
    @PrimaryKey
    public int id;
    public String imie;
    public String nazwisko;
    @Ignore
    Bitmap zdjecie;
    //AKCESORY
  
```

Atrybuty uczynić publicznymi lub przygotować akcesory (zgodnie z nazewnictwem ziarna Javy).

71

## Biblioteka Room

Encja musi definiować przynajmniej jedno pole jako klucz główny (adnotacja `@PrimaryKey`).



W celu wykorzystania mechanizmu autoinkrementacji należy ustawić atrybut `autoGenerate` adnotacji `@PrimaryKey`.



W przypadku, gdy klucz główny składa się ze zbioru atrybutów należy wykorzystać atrybut `primaryKey` adnotacji `@Entity`



```

@Entity(primaryKey={"imie","nazwisko"})
class Student{
    public String imie;
    public String nazwisko;
    ...
}
  
```

72

## Biblioteka Room

### Mapowanie nazw

Domyślnie klasa mapowana jest do tabeli o takiej samej nazwie. Aby to zmodyfikować należy wykorzystać atrybut `tableName` adnotacji `@Entity`

Domyślnie atrybuty mapowane są do kolumn o tych samych nazwach. Aby to zmienić należy wykorzystać adnotację `@ColumnInfo` przed atrybutem

```
@Entity(tableName="StudPwsz")
class Student{
    @PrimaryKey
    public int id;
    @ColumnInfo(name="nazwa")
    public String imie;
}
```

73

## Biblioteka Room

W celu utworzenia indeksu dla określonych atrybutów (w celu poprawy wydajności dostępu do danych) należy określić atrybut `indices` adnotacji `@Entity`.

```
@Entity(indices={@Index("name"),@Index(value={"imie", "n_kol"})})
class Student{
    public String imie;
    @ColumnInfo(name="n_kol")
    public String nazwisko;}

```

W celu określenia atrybutu lub zbioru atrybutów jako unikalne (wartości nie mogą się powtarzać) należy ustawić atrybut `unique` na `true` dla adnotacji `@Index`.

```
@Entity(indices={@Index(value={"i_kol", "n_kol"}, unique=true)})
class Student{
    @ColumnInfo(name="i_kol")
    public String imie;
    @ColumnInfo(name="n_kol")
    public String nazwisko;}

```

## Biblioteka Room

### Definiowanie zależności między obiektami

Zależności realizowane są poprzez ograniczenia klucza obcego pomiędzy encjami.

```
@Entity(foreignKeys= @ForeignKey(entity=Student.class,
parentColumns="id", childColumns="stud_id"))
class PracaDyplomowa{
    @PrimaryKey
    public int id;
    public String tytul;
    @ColumnInfo(name="stud_id");
    public int studId; }
```

Klucze obce umożliwiają określenie co się stanie przy aktualizacji powiązanej encji. Przykładowo określenie atrybutu `onDelete=CASCADE` w adnotacji `@ForeignKey` spowoduje, że gdy usuwany zostanie `student`, to usunięte zostaną wszystkie jego `PraceDyplomowe`.

75

## Biblioteka Room

### Kompozycja

W przypadku gdy w skład encji wchodzi inna encja, która powinna być przechowywana w tej samej tabeli należy wykorzystać adnotację `@Embedded`.

```
class Adres{
    public String ulica
    public String miasto
    @ColumnInfo(name="nr_domu")
    public int nrDomu; }

@Entity Student{
    @PrimaryKey
    public int id;
    public String Nazwisko;
    @Embedded
    public Adres adres; }
```

Kolumna `Student` będzie zawierała kolumny:  
id, Nazwisko, ulica, miasto, nr\_domu

Klasy `Embedded` mogą zawierać inne klasy `Embedded`.

76

## Biblioteka Room

W celu dostępu do danych należy wykorzystać obiekty DAO (Data Access Objects).



### Insert

Poprzedzenie metody DAO adnotacją @Insert spowoduje, że Room wygeneruje implementację metody, która zapisze wszystkie parametry do bazy danych w jednej transakcji.

```
@Dao
public interface MyDao{
    @Insert(onConflict=OnConflictStrategy.REPLACE)
    public void insertStudents(Student... students);
    @Insert
    public void insertBothStudents(Student s1, Student s2);
    @Insert
    public void insertStudentsAndTeachers(Student s,
    List<Teacher> teachers);
}
```

## Biblioteka Room

Jeżeli metoda poprzedzona @Insert przyjmuje tylko jeden parametr, zwraca typ long, który określa pole id wstawianego wiersza.



Jeżeli parametrem metody poprzedzonej adnotacją @Insert jest tablica lub kolekcja, zwraca long[] lub List<Long>.



78

## Biblioteka Room

### Update

Metoda poprzedzona adnotacją @Update modyfikuje zbiór encji przekazany w parametrze (aktualizuje wszystkie encje gdzie id wiersza jest zgodne z id parametru. Zapisuje wartości pól z parametrów do aktualizowanych wierszy.).



```
@Dao
public interface MyDao{
    @Update
    public void updateStudents(Student... student);
}
```

### Delete

Usuwa wszystkie encje przekazane w parametrze (używa klucza głównego do określenia usuwanych elementów).

```
@Dao
public interface MyDao{
    @Delete
    public void deleteStudents(Student...
    student);
}
```



## Biblioteka Room

### Zapytania

Adnotacja @Query umożliwia zdefiniowanie operacji odczytu/zapisu na bazie danych.



Przykładowo zapytanie zwracające wszystkie encje z tabeli student:

```
@Dao
public interface MyDao{
    @Query("Select * from student")
    public Student[] loadAllStudents();
}
```

### Parametry

W celu przekazania parametru do zapytania, parametr we fragmencie SQL należy poprzedzić znakiem ":". Taka sama nazwa parametru musi znajdować się w nagłówku metody:

```
@Dao
public interface MyDao{
    @Query("Select * from student where srednia > :paramSrednia")
    public Student[] loadStudents(double paramSrednia);
}
```

## Biblioteka Room

### Zwracanie podzbioru kolumn

Wybrany zbiór atrybutów, na który ma zostać wykonana projekcja zapytania musi zostać zdefiniowany w osobnej klasie (takie same nazwy atrybutów jak w encji)

```
public class Nazwa{
    @ColumnInfo(name="imie_col")
    public String imie;
    @ColumnInfo(name="nazwisko_col")
    public String nazwisko;
}

@Dao
public interface MyDao{
    @Query("Select imie_col, nazwisko_col from student")
    public List<Nazwa> loadAllNazwa();
}
```

83

## Biblioteka Room

### Przekazywanie kolekcji parametrów

Przykładowo w celu uzyskania informacji o wszystkich użytkownikach

```
@Dao
public interface MyDao{
    @Query("Select imie_col, nazwisko_col from student where
    zainteresowania in (:hobby)")
    public List<Nazwa> loadStudentsInterested(List<String>
    hobby);
}
```

82

## Biblioteka Room

### Zapytania do wielu tabel

```
@Dao
public interface MyDao{
    @Query("Select * from ksiazka INNER JOIN pozyczone on
    pozyczone.ksiazka_id=ksiazka.id INNER JOIN student ON
    student.id=pozyczone.student_id where student.nazwisko
    like :studentNazwisko")
    public List<Ksiazka> findPozyczonePrzez(String studentNazwisko)
}
```

83

```
import android.arch.persistence.room.ColumnInfo;
import android.arch.persistence.room.Entity;
import android.arch.persistence.room.ForeignKey;
import android.arch.persistence.room.PrimaryKey;

@Entity(foreignKeys=@ForeignKey(entity=Kierunek.class, parentColumns="id",
childColumns="kierunek_id"))
public class Student {
    @PrimaryKey(autoGenerate = true)
    public int id;
    @ColumnInfo(name="imie_col")
    public String imie;
    @ColumnInfo(name="nazwisko_col")
    public String nazwisko;

    public int kierunek_id;
    public String toString(){
        return ""+id+"\t"+imie+"\t"+nazwisko+"\n";
    }
}
```

```
import android.arch.persistence.room.ColumnInfo;
import android.arch.persistence.room.Entity;
import android.arch.persistence.room.PrimaryKey;

@Entity
public class Kierunek {
    @PrimaryKey(autoGenerate = true)
    public int id;
    public String nazwa;
    public int liczbaStudentow;
    public String toString(){
        return ""+id+"\t"+nazwa+"\t"+liczbaStudentow+"\n";
    }
}
```

```
import android.arch.persistence.room.Dao;
import android.arch.persistence.room.Delete;
import android.arch.persistence.room.Insert;
import android.arch.persistence.room.Query;

import java.util.List;

@Dao
public interface StudentDao {
    @Query("Select * from student")
    List<Student> getAll();
    @Query("Select * from student where kierunek_id in (:kierunekIds)")
    List<Student> loadAllInKierunek(int[] kierunekIds);
    @Query("Select * from student where imie_col like :im and nazwisko_col like :na limit 1")
    Student findByName(String im, String na);
    @Query("Select * from student inner join kierunek on student.kierunek_id=kierunek.id where kierunek.nazwa=:kier")
    public List<Student> findStudOnKier(String kier);
    @Insert
    void insertAll(Student... students);
    @Insert
    void insertAll(Kierunek... kierunek);
    @Delete
    void delete(Student... studentns);
}
```

```
package pl.nysa.pwsz.db;

import android.arch.persistence.room.Database;
import android.arch.persistence.room.RoomDatabase;

@Database(entities={Student.class, Kierunek.class}, version=1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract StudentDao studentDao();
}
```

```
public class MainActivity extends Activity implements Runnable {
    String dane="";
    public void run()
    {
        . . .
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Thread tr = new Thread(this);
        tr.start();
        Button b =(Button) findViewById(R.id.button);
        b.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                TextView t= (TextView) findViewById(R.id.textView);
                t.setText(dane);
            }
        });
    }
}
```

```

AppDatabase db = Room.databaseBuilder(getApplicationContext(),
AppDatabase.class, "nazwa_bazy").build();
StudentDao interf = db.studentDao();
Kierunek k1= new Kierunek(); k1.liczbaStudentow=5; k1.nazwa="informatyka";
Kierunek k2= new Kierunek(); k2.liczbaStudentow=3; k2.nazwa="zarzadzanie";
Student s1= new Student(); s1.imie="Jan"; s1.nazwisko="Kowalski";
s1.kierunek_id=1;
Student s2= new Student(); s2.imie="Anna"; s2.nazwisko="Malinowska";
s2.kierunek_id=1;
Student s3= new Student(); s3.imie="Marek"; s3.nazwisko="Nowak";
s3.kierunek_id=2;
Student s4= new Student(); s4.imie="Zenon"; s4.nazwisko="Nowak";
s4.kierunek_id=2;
interf.insertAll(k1, k2);
interf.insertAll(s1,s2,s3,s4);
List<Student> lista = interf.getAll();
for(Student x : lista){dane+=x.toString(); }
dane+="\n";
lista= interf.loadAllInKierunek(new int[]{1,3,5,7});
for(Student x : lista){ dane+=x.toString(); }
dane+="\n";
Student z= interf.findByName("Jan","Kowalski");
dane+=z.toString();
dane+="\n";
lista= interf.findStudOnKier("informatyka");
for(Student x : lista){ dane+=x.toString(); }

```

Po trzykrotnym przyciśnięciu przycisku:

BUTTON

1 Jan Kowalski  
2 Anna Malinowska  
3 Marek Nowak  
4 Zenon Nowak  
5 Jan Kowalski  
6 Anna Malinowska  
7 Marek Nowak  
8 Zenon Nowak  
9 Jan Kowalski  
10 Anna Malinowska  
11 Marek Nowak  
12 Zenon Nowak

1 Jan Kowalski  
2 Anna Malinowska  
5 Jan Kowalski  
6 Anna Malinowska  
9 Jan Kowalski  
10 Anna Malinowska

1 Jan Kowalski

1 Jan Kowalski  
1 Anna Malinowska  
1 Jan Kowalski  
1 Anna Malinowska  
1 Jan Kowalski  
1 Anna Malinowska