Przygotowanie aplikacji

1. Uruchom projekt z lab4. Jeżeli go nie posiadasz na dysku lokalnym skorzystaj z repozytorium GitHub (File->New->Project from version control->GitHub)
2. Zaktualizuj lokalny projekt z projektem na serwerze GitHub (VCS->Update Project)
3. Uruchom aplikację i sprawdź, czy działa poprawnie

Opis aplikacji:

Chcemy utworzyć aplikację, która przechowuje dane zobaczonych zwierząt.

W tym celu zmodyfikujemy odpowiednio poprzednią aplikację:

- *Lista będzie zawierać skomplikowaną strukturę danych*
- *Aplikacja będzie korzystać z bazy danych*

Tworzenie bazy danych i metod CRUD

1. Utwórz klasę o nazwie Animal (File->new->javaClass)
2. Dodaj do klasy interfejs „Serializable”
3. Utwórz zmienne typu private przechowujące informacje o danym zwierzęciu
    a. _id (int) (z podkreślnikiem)
    b. gatunek (String)
    c. kolor (String)
    d. wielkość (float)
    e. opis (String)
4. Utwórz pusty konstruktor danej klasy (zawsze musi być)
5. Utwórz konstruktor danej klasy, który przyjmie 4 parametry i uzupełni zmienne prywatne tworzonego obiektu
6. Przeciąż publiczną metodę zwracającą String o nazwie „toString()” zwracając tekst, zawierający parametry zwierzęcia (bez opisu)
7. Dodaj „getery” (metody zwracające poszczególne pola obiektu)
8. Utwórz nową klasę „MySQLite” (file -> new -> javaClass)
9. Przy nazwie klasy dodaj jej rozszerzenie:

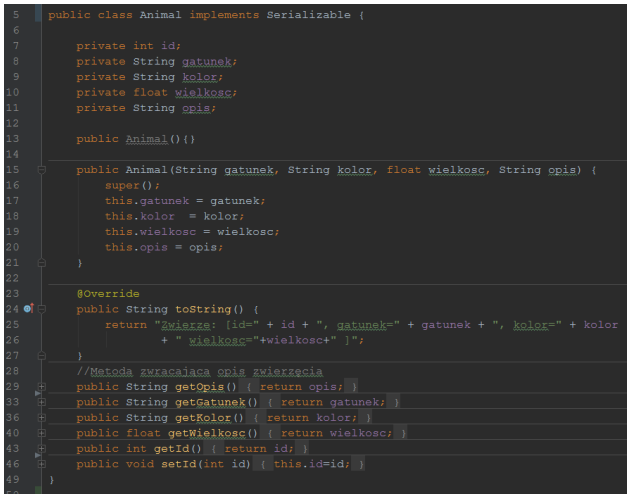
```
public class MySQLite extends
SQLiteOpenHelper {
```

10. Dodaj zmienną:

```
private static final int
DATABASE_VERSION = 1;
```

11. Utwórz konstruktor danej klasy:

```
public MySQLite(Context context) {
    super(context, "animalsDB",
```



```
public class Animal implements Serializable {

    private int id;
    private String gatunek;
    private String kolor;
    private float wielkosc;
    private String opis;

    public Animal(){}

    public Animal(String gatunek, String kolor, float wielkosc, String opis) {
        super();
        this.gatunek = gatunek;
        this.kolor  = kolor;
        this.wielkosc = wielkosc;
        this.opis = opis;
    }

    @Override
    public String toString() {
        return "Zwierze: [id=" + id + ", gatunek=" + gatunek + ", kolor=" + kolor
                + " wielkosc="+wielkosc+" ]";
    }
    //Metoda zwracająca opis zwierzęcia
    public String getOpis() { return opis; }
    public String getGatunek() { return gatunek; }
    public String getKolor() { return kolor; }
    public float getWielkosc() { return wielkosc; }
    public int getId() { return id; }
    public void setId(int id) { this.id=id; }
}
```

Baza SQLite ma wiele ograniczeń w stosunku do innych popularnych baz wykorzystujących język zapytań SQL.

Podstawowe konstrukcje języka SQL mają tą samą składnię.

W SQLite mamy tylko 5 typów danych:

- *Text (string w kodowaniu UTF)*
- *Integer (liczby całkowite)*
- *Real (liczby zmiennoprzecinkowe zapisane na 8 bajtach)*
- *Blob (zapis bitowy-zapisane tak, jak zostały wprowadzone)*
- *Null (pusty typ)*

```
null, DATABASE_VERSION);
}
```

Przy kopiowaniu kodów proszę zwrócić uwagę na podkreślnik przy _id (czasami go zjada, bądź jest niewidoczny)

12. Przeciąż metodę onCreate i onUpgrade:

```
@Override
public void onCreate(SQLiteDatabase
database) {
    String DATABASE_CREATE =
"create table animals " +
 "(_id integer primary key autoincrement," +
     "gatunek text not null," +
     "kolor text not null," +
     "wielkosc real not null," +
     "opis text not null);";
    database.execSQL(DATABASE_CREATE);
}

@Override
 public void onUpgrade(SQLiteDatabase db,
      int oldVersion, int newVersion) {
  db.execSQL("DROP TABLE IF EXISTS
animals");
  onCreate(db);
}
```

W metodzie onCreate tworzymy nową bazę danych SQLite o zadanej strukturze

W metodzie onUpgrade resetujemy bazę danych, np. gdy dodamy nowe pola do tabeli – usunięcie starej i wszystkich danych jakie zawiera

13. Dodaj metodę dodającą nowy wpis do bazy danych:

```
public void dodaj(Animal zwierz){

    SQLiteDatabase db =
this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put("gatunek", zwierz.getGatunek());
    values.put("kolor", zwierz.getKolor());
    values.put("wielkosc", zwierz.getWielkosc());
    values.put("opis", zwierz.getOpis());

    db.insert("animals", null, values);

    db.close();
}
```

Aby dodać elementy do bazy danych należy:

1. *Utworzyć obiekt SQLiteDatabase*
2. *Utworzyć obiekt ContentValues, który należy zapełnić strukturą zgodną z tabelą, do której wprowadzamy dane*
3. *Uruchomić metodę .insert() podając jako parametry tabelę do której dodajemy obiekt typu ContentValues*

14. Dodaj metodę usuwającą wpis z bazy danych:

```
public void usun(String id) {
    SQLiteDatabase db =
this.getWritableDatabase();
    db.delete("animals", "_id = ?",
  new String[] { id });
    db.close();
}
```

Usuwanie wpisu z DB realizujemy metodą .delete() z parametrami takimi jak nazwa tabeli, warunek (który element ma być usunięty), i parametr warunku

15. Dodaj metodę modyfikującą wpis w bazie danych:

```
public int aktualizuj(Animal zwierz) {

    SQLiteDatabase db =
this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put("gatunek", zwierz.getGatunek());
    values.put("kolor", zwierz.getKolor());
    values.put("wielkosc", zwierz.getWielkosc());
    values.put("opis", zwierz.getOpis());

    int i = db.update("animals", values, "_id =
?", new String[]{String.valueOf(zwierz.getId())});

    db.close();

    return i;
}
```

Aktualizacja przebiega podobnie jak dodawanie wpisu, z tym że wprowadzamy warunek (jaki element ma być modyfikowany)

16. Dodaj metodę pobierającą wpis z bazy danych:

```
public Animal pobierz(int id){

    SQLiteDatabase db =
this.getReadableDatabase();

    Cursor cursor =
            db.query("animals", //a. table name
                    new String[] { "_id",
"gatunek", "kolor", "wielkosc", "opis" }, // b.
column names
                    "_id = ?", // c. selections
                    new String[] {
String.valueOf(id) }, // d. selections args
                    null, // e. group by
                    null, // f. having
                    null, // g. order by
                    null); // h. limit

    if (cursor != null)
        cursor.moveToFirst();

    Animal zwierz = new
Animal(cursor.getString(1), cursor.getString(2),
cursor.getFloat(3), cursor.getString(4));

zwierz.setId(Integer.parseInt(cursor.getString(0))
);

    return zwierz;
}
```

Cursor jest obiektem typu iterator, który „przechodzi" po kolejnych elementach w bazie danych.

Obiekt Cursor przyjmuje jako parametr nazwę tabeli, tabelę nazw poszczególnych pól (kolumn), warunki itp.

Aby pobrać dane z wiersza na który wskazuje Cursor należy skorzystać z wbudowanych metod getString, getFloat itp. (w zależności od typu danych)

17. Dodaj metodę zwracającą Cursor do bazy danych:

```
public Cursor lista(){
  SQLiteDatabase db =
        this.getReadableDatabase();
   return db.rawQuery("Select * from
         animals",null);
}
```

Metoda Lista zwraca obiekt Cursora dla SimpleCursorAdapter, w danym przykładzie użyto pełnego zapytania SQL a nie wbudowanych funkcji

18. Wykonaj Commita wprowadzonych zmian

Wyświetlanie wpisów

1. W pliku MainActivity.java należy zmienić rodzaj adaptera przypisanego do ListView z ArrayAdapter na SimpleCursorAdapter
2. Utwórz obiekt „db" utworzonej wcześniej klasy MySQLite
3. Następnie w metodzie onCreate zmieniamy sposób inicjalizacji adaptera:

```
this.adapter = new SimpleCursorAdapter(
        this,
        android.R.layout.simple_list_item_2,
        db.lista(),
        new String[] {"_id", "gatunek"},
        new int[] {android.R.id.text1,
android.R.id.text2},

SimpleCursorAdapter.IGNORE_ITEM_VIEW_TYPE
    );
```

Adapter SimpleCursorAdapter pozwala na wyświetlanie elementów listy od dowolnego dostawcy danych. W niniejszym przykładzie wewnętrznej bazie danych SQLite

Jako parametry podajemy

- *Context (aktywne okno aplikacji)*
- *Wyświetlany typ (simple_list_item_2 oznacza, że dla kontrolki list item zostanie wyświetlony ciąg złożony z 2 zmiennych text1 i text2*
- *Cursor którym adapter nawiguje po zbiorze danych*
- *Pola z bazy danych które będą wyświetlane*
- *Identyfikatory zmiennych do których będą zapisane pola od dostawcy treści*
- *Flaga adaptera*

4. Uruchom aplikację (powinien być pusty ekran, jako że baza danych jest pusta)
5. Wykonaj Commita

Dodawanie wpisów

1. Zmodyfikuj formularz activity_dodaj_wpis wprowadzając dodatkowe pola opisujące zwierzę (takie jak w klasie Animal)
2. W pliku DodajWpis.java w metodzie onCreate utwórz ArrayAdapter zawierający listę dostępnych do wyboru gatunków i przypisz go do spinera

```
ArrayAdapter gatunki = new ArrayAdapter(
   this,
android.R.layout.simple_spinner_dropdown_item,
   new String[] {"Pies", "Kot", "Rybki"});
Spinner gatunek = (Spinner) findViewById
(R.id.gatunek);
gatunek.setAdapter(gatunki);
```

Pole gatunek powinno być rozwijaną listą do wyboru

Wielkość powinno być polem liczbowym

Opis powinno mieć kilka linijek

Aby dodać etykietę do pola użyj parametru android:hint (brak w spinerze)

Tworząc adapter musimy podać jakim typem będzie zwracana wartość. Z bogatej kolekcji możliwości za pomocą autouzupełniania znajdujemy typ dedykowany spinerowi (simple_spinner_dropdown_item)

3. Zmodyfikuj metodę „wyślij” w DodajWpis.java
   - Pobierz wartości poszczególnych pól
   - Utwórz obiekt typu Animal
   - Utwórz intencję i dodaj do niej dane „Extra” – utworzony obiekt typu Animal
   - Ustaw zwracanie danych z Activity
   - Zakończ działanie danego Activity

```
public void wyslij (View view)
{
    EditText kolor = (EditText) findViewById
(R.id.kolor);
    EditText wielkosc = (EditText)
findViewById (R.id.wielkosc);
    EditText opis = (EditText) findViewById
(R.id.opis);
    Spinner gatunek = (Spinner) findViewById
(R.id.gatunek);

    Animal zwierze = new Animal(
  gatunek.getSelectedItem().toString(),
  kolor.getText().toString(),
Float.valueOf(wielkosc.getText().toString()),
  opis.getText().toString()
    );
    Intent intencja = new Intent();
    intencja.putExtra("nowy", zwierze);
    setResult(RESULT_OK, intencja);
    finish();
}
```

4. Zmodyfikuj metodę onActivityResult w pliku MainActivity.java odpowiedzialną za odbiór danych z formularza zmieniając typ pobieranych danych:

```
Animal nowy = (Animal)
extras.getSerializable("nowy");
```

5. Dodaj obiekt „nowy” do bazy danych:

```
this.db.dodaj(nowy);
```

6. Zmień sposób odświeżania listy:

```
adapter.changeCursor(db.lista());
adapter.notifyDataSetChanged();
```

W odróżnieniu od programu z Lab4 dane przechowywane są w bazie danych. Dzięki temu wyłączenie aplikacji i jej ponowne uruchomienie nie traci wprowadzonych danych.

7. Przetestuj dodawanie wpisów do bazy danych
8. Wykonaj Commita

Edycja wpisów

1. W pliku MainActivity.java w metodzie onCreate utwórz listner, który będzie reagować na naciśnięcie pojedynczego pola z listView:

```
listview.setOnItemClickListener(new
AdapterView.OnItemClickListener()
```

Listner jest jedną z dwóch metod dodawania aktywności do przycisku/obiektu.

Utworzenie listnera wymaga utworzenia obiektu z przeciążeniem wewnętrznej metody –

```
{
    @Override
    public void onItemClick(AdapterView<?>
adapter, View view,int pos, long id)
    {

    }
});
```

w zależności od rodzaju nasłuchiwanej akcji.

onItemClick przyjmuje prócz adaptera wiersz, który został kliknięty, jego pozycję, oraz id (zgodny z tym w obiekcie R.id)

2. Wewnątrz utworzonego listnera, w metodzie onItemClick pobierz wartość zmiennej id (zachowana w R.id.text1):

```
TextView name = (TextView)
view.findViewById(android.R.id.text1);
```

Android wymaga częstej konwersji typu zmiennych. W tym przypadku dane musimy pobrać jako string a następnie przekonwertować do typu int

3. Utwórz obiekt typu Animal pobierając z bazy danych wartości danego wpisu:

```
Animal zwierz = db.pobierz(Integer.parseInt
(name.getText().toString()));
```

4. Stwórz intencję, dodaj nowoutworzony obiekt Animal jako „Extra” i uruchom formularz z innym kodem zwrotu:

```
Intent intencja = new
Intent(getApplicationContext(),
DodajWpis.class);
intencja.putExtra("element", zwierz);
startActivityForResult(intencja, 2);
```

5. W pliku DodajWpis.java utwórz prywatną zmienną „modyfi_id”
6. W metodzie onCreate po utworzeniu formularza stwórz obiekt Bundle, który odbierze dane Extra wysłane do aktywności
7. W bloku try{}catch{} sprawdź czy istnieje przesłany obiekt, jeżeli nie – ustaw modyfi_id na 0
8. Gdy istnieje przesłany obiekt pobierz dane extra jako obiekt typu Animal
9. Zmapuj pola formularza
10. Do każdego pola formularza przypisz wartość odpowiedniej zmiennej przesłanego obiektu
11. Ustaw modyfi_id na id przesłanego obiektu

```
Bundle extras = getIntent().getExtras();
try {
    if(extras.getSerializable(„element”) !=
null) {
        Animal zwierz = (Animal)
extras.getSerializable(„element”);

        EditText kolor = (EditText)
findViewById(R.id.kolor);
        EditText 5ielkość = (EditText)
findViewById(R.id.wielkosc);
        EditText opis = (EditText)
findViewById(R.id.opis);

        kolor.setText(zwierz.getKolor());
        5ielkość.setText(
Float.toString(zwierz.getWielkosc()));
        opis.setText(zwierz.getOpis());

        this.modyfi_id=zwierz.getId();
    }
}catch(Exception ex) {
    this.modyfi_id=0;
}
```

Jeżeli użytkownik modyfikuje obiekt, zwrócony przez formularz obiekt będzie posiadał niezerowy

12. W metodzie „wyślij” przed utworzeniem intencji dodaj przypisanie id do zwracanego obiektu:

```
zwierze.setId(this.modyfi_id);
```

13. W pliku MainActivity.java do metody onActivityResult należy dodać obsługę powrotu z kodu „2” – edycji, która wygląda dokładnie tak samo jak dodanie wpisu, z różnicą wywołania funkcji db.dodaj() na db.aktualizuj()

W przykładzie nie ustawiamy wartości Spinnera, gdyż wymaga znajomości pozycji a nie wartości wybranej opcji

14. Sprawdź poprawność dodawania i edycji wpisów
15. Przeprowadź commit

Usuwanie wpisów

1. W metodzie onCreate pliku MainActivity.java dodaj listner onItemLongClick:

```
listview.setOnItemLongClickListener(new
AdapterView.OnItemLongClickListener() {
    @Override
    public boolean
onItemLongClick(AdapterView<?> parent, View
view, int position, long id) {
        return true;
    }
});
```

IDE AndroidStudio ułatwia tworzenie Listnerów. Korzystając z autouzupełniania przy ich tworzeniu powstaje cała struktura, a nie tylko nazwa.

2. Pobierz id obiektu Animal kliknięty wpisu (tak jak w listnerze onItemClick)
3. Wywołaj metodę db.usun()
4. Zaktualizuj adapter (tak jak w metodzie onActivityResult
5. Sprawdź czy przytrzymanie wpisu na ListView go usuwa
6. Przeprowadź Commita oraz Push

Pisząc aplikacje, każde usunięcie danych musi być dodatkowo potwierdzane przez użytkownika (dwuetapowo), szczególnie jeżeli dane są bezpowrotnie tracone