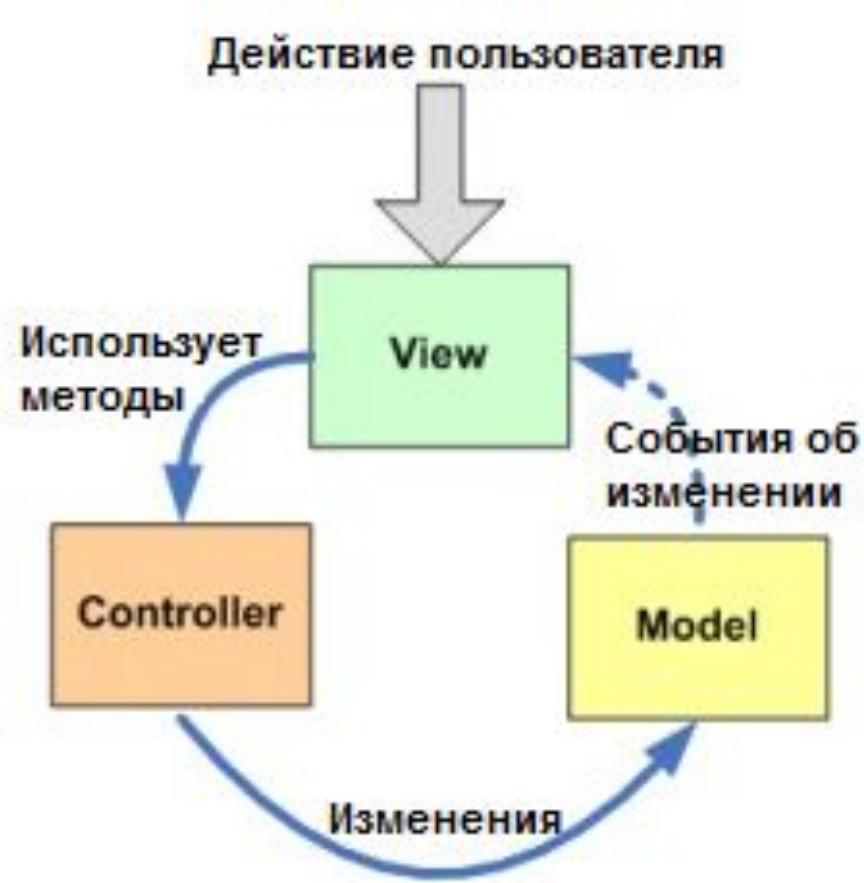
Архитектура андроид приложений
Паттерны MVC, MVP.

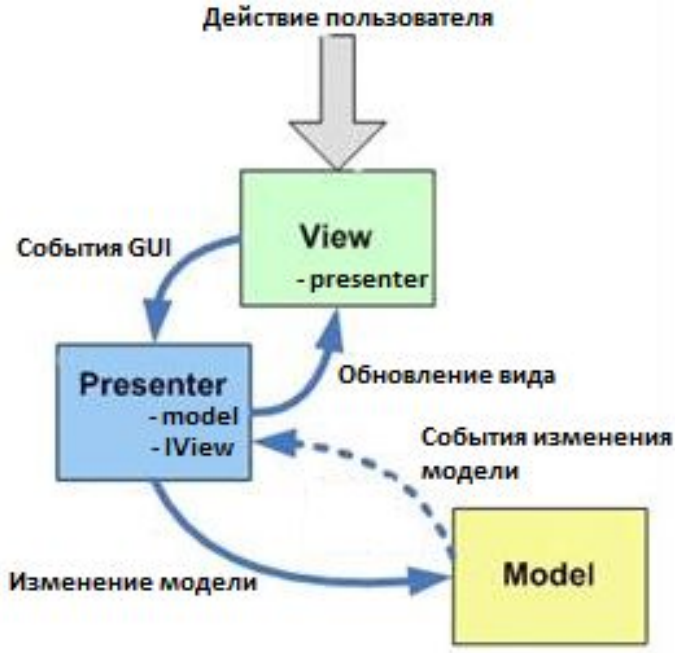
https://www.fandroid.info/lecture-5-on-the-architecture-of-the-android-application-mvp-pattern/
http://www.ohandroid.com/mvc-android.html
https://upday.github.io/blog/model-view-presenter/
https://ichi.pro/ru/patterny-arhitektury-android-cast-1-model-predstavlenie-kontroller-65621982573238

Паттерн MVC


Действие пользователя
View
Использует
методы
События об
изменении
Controller
Model
Изменения

Паттерн MVP

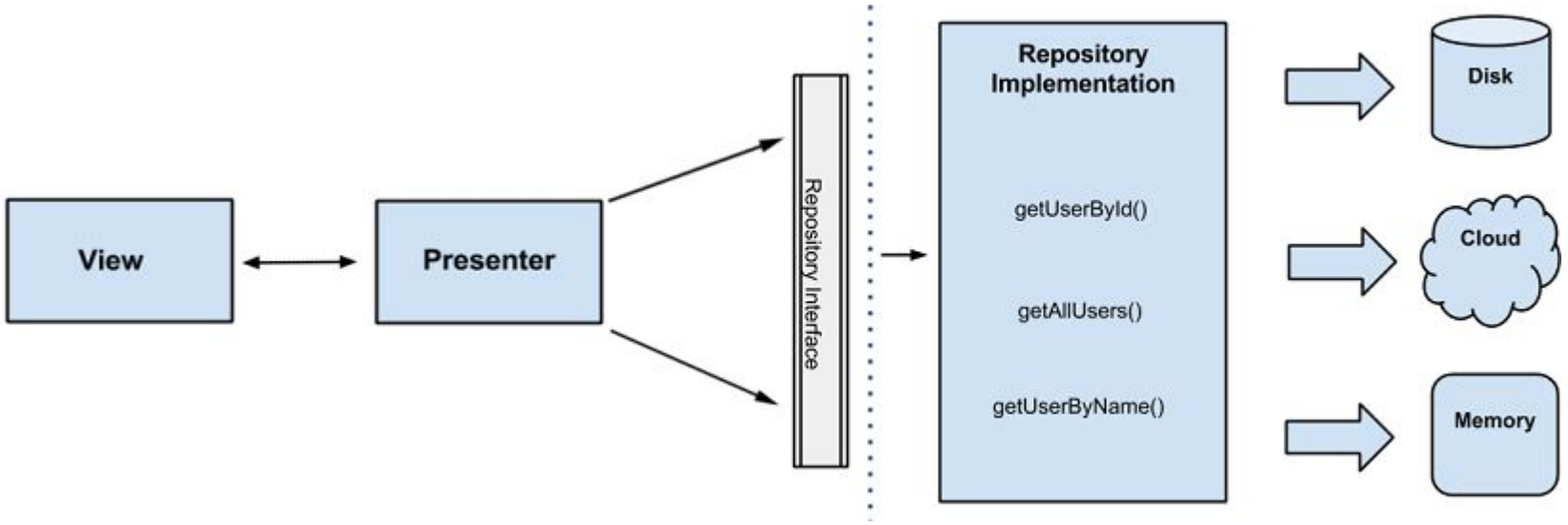
Model-View-Presenter



Паттерн MVP нашел большее применение в Android.
MVP имеет несколько основных отличий от MVC:

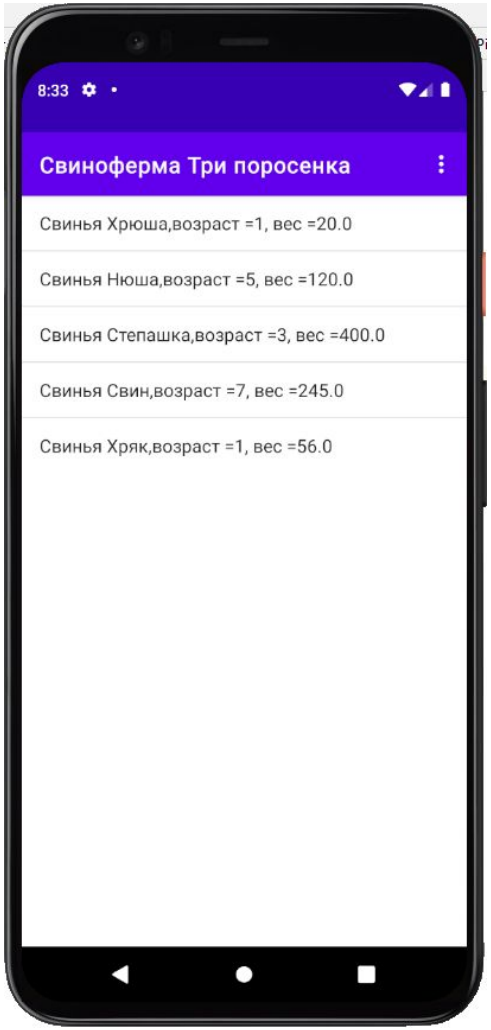
- Presenter управляет только одной View и взаимодействует с ней через специальный интерфейс.
- View управляется только с помощью Presenter-а, а не отслеживает изменение Model.
- Presenter получает все данные из Model (или из слоя данных), обрабатывает их в соответствии с требуемой логикой и управляет View.

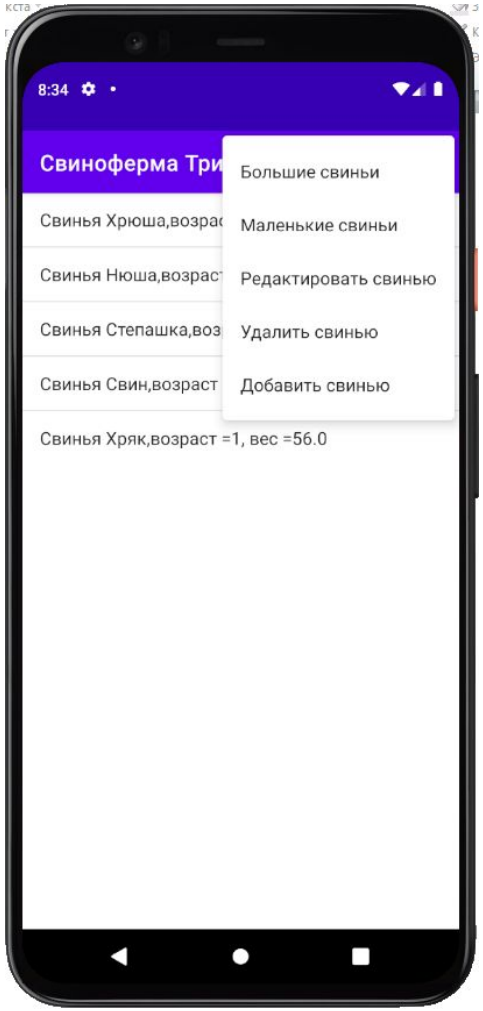
- View знает о Presenter;
- Presenter знает о View и Model (Repository);
- Model сама по себе;

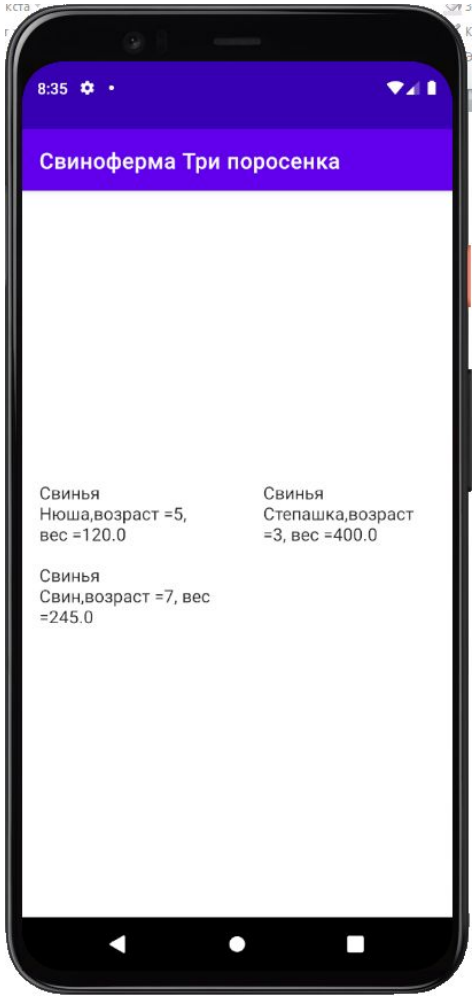


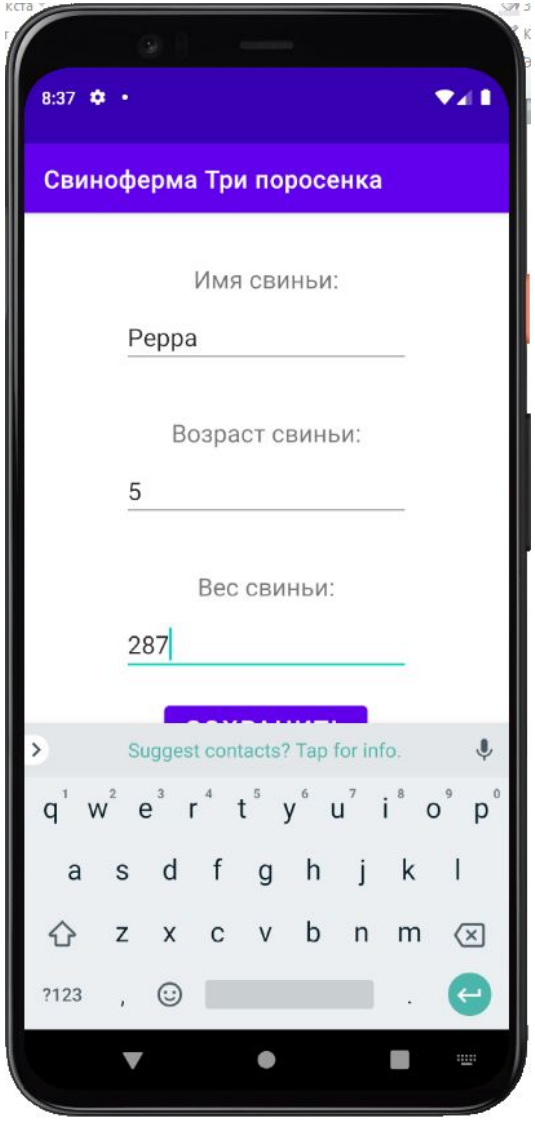
Пример. Разработать Android приложение согласно варианту с использованием архитектуры «***Модель-Представление-Презентер»***, выполняющее ввод данных, вывод и редактирование в соответствии с вариантом. Для выполнения каждого пункта задания использовать отдельную Activity и модель. Организовать навигацию с использованием списковых представлений (ListView), при этом при помощи адаптера осуществить связывание с массивами данных.

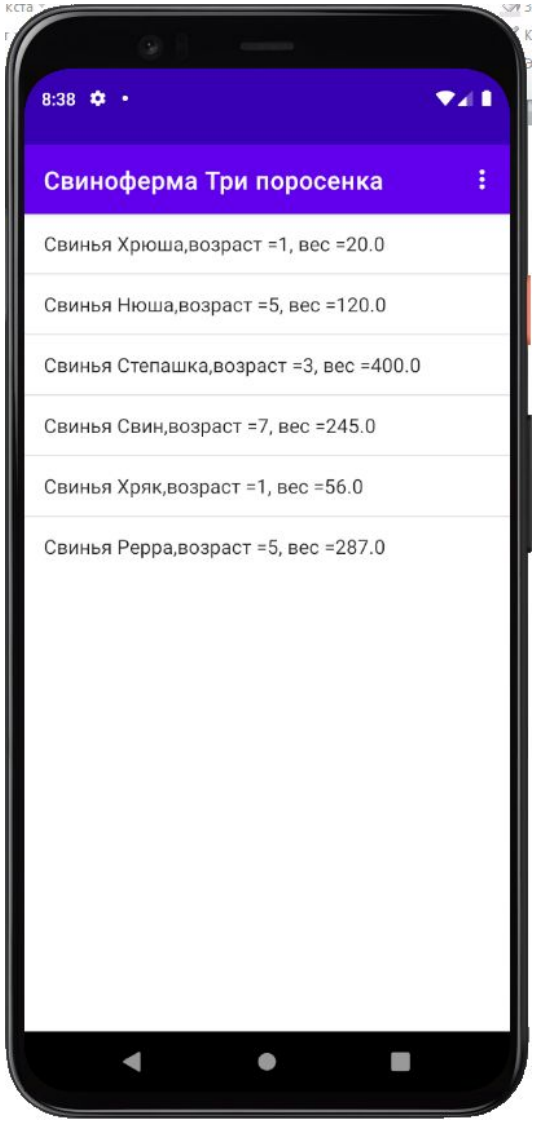
Для выбора действия использовать меню.

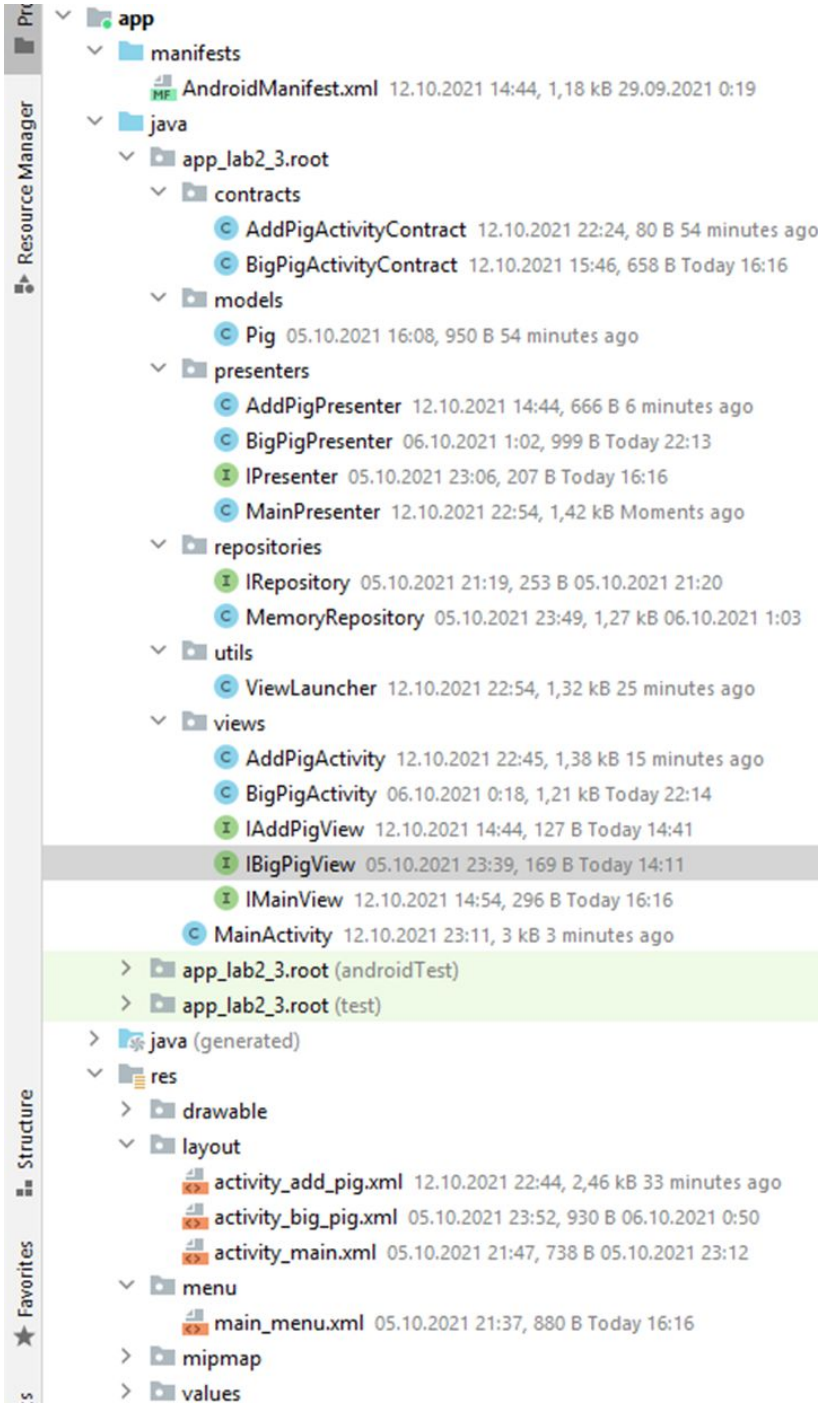

8:33
Свиноферма Три поросенка
Свинья Хрюша,возраст =1, вес =20.0
Свинья Нюша,возраст =5, вес =120.0
Свинья Степашка,возраст =3, вес =400.0
Свинья Свин,возраст =7, вес =245.0
Свинья Хряк,возраст =1, вес =56.0


8:34
Свиноферма Три
Большие свиньи
Маленькие свиньи
Редактировать свинью
Удалить свинью
Добавить свинью
Свинья Свин,возраст
Свинья Хряк,возраст =1, вес =56.0


8:35
Свиноферма Три поросенка
Свинья Нюша,возраст =5, вес =120.0
Свинья Степашка,возраст =3, вес =400.0
Свинья Свин,возраст =7, вес =245.0


8:37
Свиноферма Три поросенка
Имя свиньи:
Peppa
Возраст свиньи:
5
Вес свиньи:
287
Suggest contacts? Tap for info.


8:38
Свиноферма Три поросенка
Свинья Хрюша,возраст =1, вес =20.0
Свинья Нюша,возраст =5, вес =120.0
Свинья Степашка,возраст =3, вес =400.0
Свинья Свин,возраст =7, вес =245.0
Свинья Хряк,возраст =1, вес =56.0
Свинья Peppa,возраст =5, вес =287.0




```java
public interface IRepository {
    Pig get(String name);
    ArrayList<Pig> getAll();
    void add(Pig pig);
    void delete(Pig pig);
}
```

```
public class MemoryRepository implements IRepository {
    private ArrayList<Pig> pigs = new ArrayList<Pig>();
private static IRepository repository;
    private MemoryRepository() {
        pigs.add(new Pig("Хрюша", 1, 20));
        pigs.add(new Pig("Нюша", 5, 120));
        pigs.add(new Pig("Степашка", 3, 400));
        pigs.add(new Pig("Свин", 7, 245));
        pigs.add(new Pig("Хряк", 1, 56));
    }
    public static IRepository getRepository() {
        if(repository==null)
            repository=new MemoryRepository();
        return repository;
    }
    @RequiresApi(api = Build.VERSION_CODES.N)
    @Override
    public Pig get(String name) {
        return pigs.stream().filter(p->p.getName()
                .equals(name)).findFirst().get();
    }
    @Override
    public ArrayList<Pig> getAll() {        return pigs;    }
    @Override
    public void add(Pig pig) {       pigs.add(pig);    }
    @Override
    public void delete(Pig pig) {        pigs.remove(pig);    }
}
```

```
public interface IMainView {
    void showPigs(List<Pig> pigs);
    void showEditPig(Pig pig);
    void deletePigs();
    void showBigPigs();
    void showLittlePigs();
    void addPig();
}
```

```
public interface IPresenter {
    void onStart();
    void onBigPigClicked();
    void onlittlePigClicked();
    void onAddPigClicked();
    void onDestroy();
}
```

```
public class MainPresenter implements IPresenter {
    //Компоненты MVP приложения
    private IMainView mView;
    private IRepository mRepository;
     private ViewLauncher launcher; //объект для старта активностей

    public MainPresenter(IMainView mView) {
        this.mView = mView;
        this.mRepository = MemoryRepository.getRepository();
        launcher=new ViewLauncher((AppCompatActivity)mView);
    }
```

```
//при старте View
@Override
public void onStart() {
    mView.showPigs(mRepository.getAll());
}
//View сообщает, что был выбрана команда "Большие свиньи"
@Override
public void onBigPigClicked() {
    //mView.showBigPigs();
    launcher.startBigPigActivity();
}
```

```
//View сообщает, что был выбрана команда "Добавить свинью"
@Override
public void onAddPigClicked() {
    launcher.startAddPigActivity();
    //mView.addPig();
}

И т.д.
}
```

```
public class MainActivity extends AppCompatActivity implements IMainView {
    private IPresenter mPresenter;
    private ListView pigsList;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        pigsList = (ListView) findViewById(R.id.pigsList);
        //Создаём Presenter и в аргументе передаём
        mPresenter = new MainPresenter(this);
        mPresenter.onStart();
    }

    @Override
    protected void onResume() {
        super.onResume();
        mPresenter.onStart();
    }
```

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {

    getMenuInflater().inflate(R.menu.main_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    int id = item.getItemId();

    switch (id) {
        case R.id.bigPigs:
            mPresenter.onBigPigClicked();
            return true;
        case R.id.addPig:
            mPresenter.onAddPigClicked();
            return true;
        case 2:

            return true;
    }
    return super.onOptionsItemSelected(item);
}
```

```
public void showPigs(List<Pig> pigs) {

    // создаем адаптер
    ArrayAdapter<Pig> adapter = new ArrayAdapter(this,
            android.R.layout.simple_list_item_1, pigs);
    // устанавливаем для списка адаптер
    pigsList.setAdapter(adapter);
}
```

И т.д.

```
public class BigPigPresenter {
    //Компоненты MVP приложения
    private IBigPigView view;
    private IRepository repository;

    public BigPigPresenter(IBigPigView view) {
        this.view = view;
        repository= MemoryRepository.getRepository();
    }

    @RequiresApi(api = Build.VERSION_CODES.N)
    public void onStart() {
        ArrayList<Pig> bigPigs= (ArrayList<Pig>) repository.getAll().stream()
                .filter(p->p.getWeight()>100).collect(Collectors.toList());
       view.showPigs(bigPigs);

    }
}
```

```
public interface IBigPigView
{
    void showPigs(List<Pig>
pigs);
}
```

```
public class BigPigActivity extends AppCompatActivity implements IBigPigView {
    private BigPigPresenter presenter;
    @RequiresApi(api = Build.VERSION_CODES.N)
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_big_pig);
        presenter=new BigPigPresenter(this);
        presenter.onStart();
    }

    @Override
    public void showPigs(List<Pig> pigs) {
        // получаем элемент GridView
        GridView pigsList = (GridView) findViewById(R.id.gridview);
        // создаем адаптер
        ArrayAdapter<Pig> adapter = new ArrayAdapter<Pig>(this,
                android.R.layout.simple_list_item_1, pigs);
        pigsList.setAdapter(adapter);
    }
}
```

```
public class AddPigPresenter {
    //Компоненты MVP приложения
    private IAddPigView view;
    private IRepository repository;

    public AddPigPresenter(IAddPigView view) {
        this.view = view;
        repository= MemoryRepository.getRepository();
    }
    public void addPig()
    {
        Pig pig=view.getData();
        repository.add(pig);
    }
}
```

```
public interface IAddPigView {
    Pig getData();
}
```

```
public class AddPigActivity extends AppCompatActivity implements  IAddPigView {
    private AddPigPresenter presenter;
    private EditText nameEditText;
    private EditText ageEditText;
    private EditText weightEditText;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_add_pig);
        presenter=new AddPigPresenter(this);
        nameEditText=(EditText)findViewById(R.id.name);
        ageEditText=(EditText)findViewById(R.id.age);
        weightEditText=(EditText)findViewById(R.id.weight);
    }
    @Override
    public Pig getData() {

        return new Pig(nameEditText.getText().toString(),
                Integer.valueOf(ageEditText.getText().toString()),
                Double.valueOf(weightEditText.getText().toString()));
    }
    public void savePig(View view) {
        presenter.addPig();
        //возвращаемся назад
        this.onBackPressed();
    }
}
```

```
public class ViewLauncher {

    private ActivityResultLauncher bigPiglauncher;
    private ActivityResultContract contract;
    private AppCompatActivity view;

    public ViewLauncher(AppCompatActivity view) {
        this.view = view;

        //регистрация контракта активности
        bigPiglauncher = view.registerForActivityResult(new BigPigActivityContract(),

                new ActivityResultCallback<Object>() {
                    @Override
                    public void onActivityResult(Object result) {
                        // обработка результата
                    }
                });
    }
    public void startBigPigActivity(){
        bigPiglauncher.launch(null);
    }
    public void startAddPigActivity(){
        view.startActivity(new Intent(view.getApplicationContext(), AddPigActivity.class));
    }
}
```

```
public class BigPigActivityContract extends ActivityResultContract {
    @NonNull
    @Override
    public Intent createIntent(@NonNull Context context, Object input) {
        return new Intent(context, BigPigActivity.class);
    }

    @Override
    public Object parseResult(int resultCode, @Nullable Intent intent) {
        return null;
    }
}
```

Создание контекстного меню

***Контекстное меню** вызывается в **Андроид** длительным нажатием на каком-либо экранном компоненте. Обычно оно используется в списках, когда на экран выводится список однородных объектов*

В контекстном меню не поддерживаются значки и быстрые клавиши. Контекстное меню применимо к **View**, а меню к **Activity**. Поэтому в приложении может быть ***одно*** меню и несколько контекстных меню, например у каждого элемента TextView.

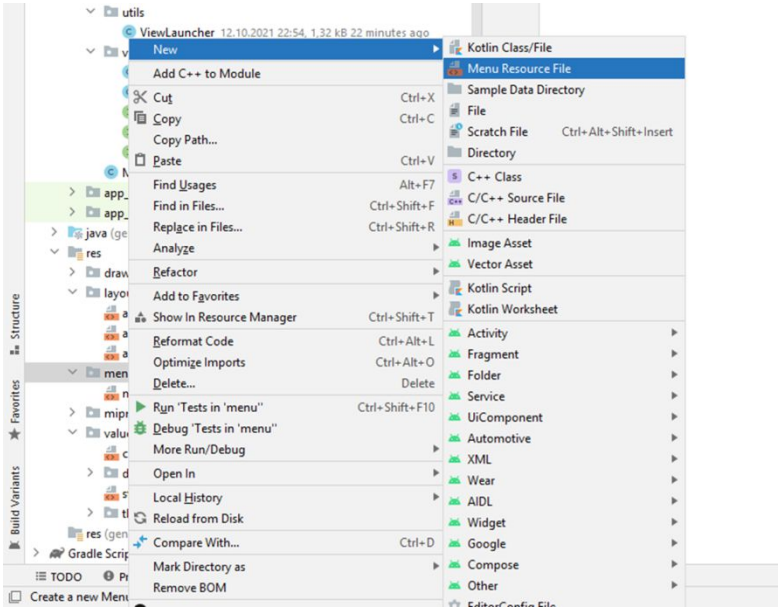
Существует два способа предоставления возможности контекстных действий:

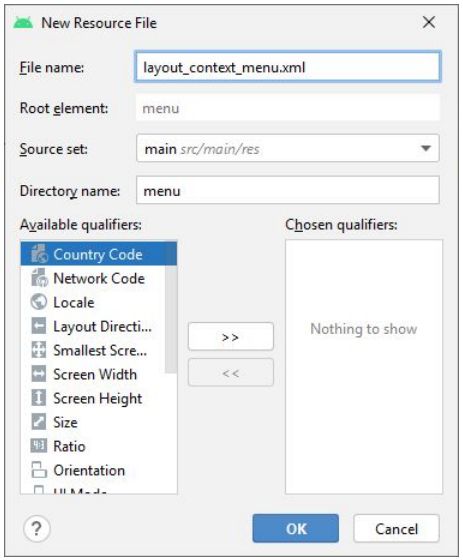
- в плавающем контекстном меню. При этом меню отображается в виде плавающего списка пунктов меню (похоже на диалоговое окно). Пользователи могут каждый раз выполнять контекстное действие только с одним элементом

- В режиме контекстных действий. Этот режим является системной реализацией ActionMode, которая отображает строку контекстных действий вверху экрана с пунктами действий, которые затрагивают выбранные элементы.

Когда этот режим активен, пользователи могут одновременно выполнять действие с несколькими элементами. Если данный режим предусмотрен, именно его рекомендуется использовать для отображения контекстных действий.

Создание плавающего контекстного меню:





- Зарегистрировать все элементы для контекстного меню.

```
nameEditText=(EditText)findViewById(R.id.name);
ageEditText=(EditText)findViewById(R.id.age);
weightEditText=(EditText)findViewById(R.id.weight);


this.registerForContextMenu(this.nameEditText);
this.registerForContextMenu(this.ageEditText);
```

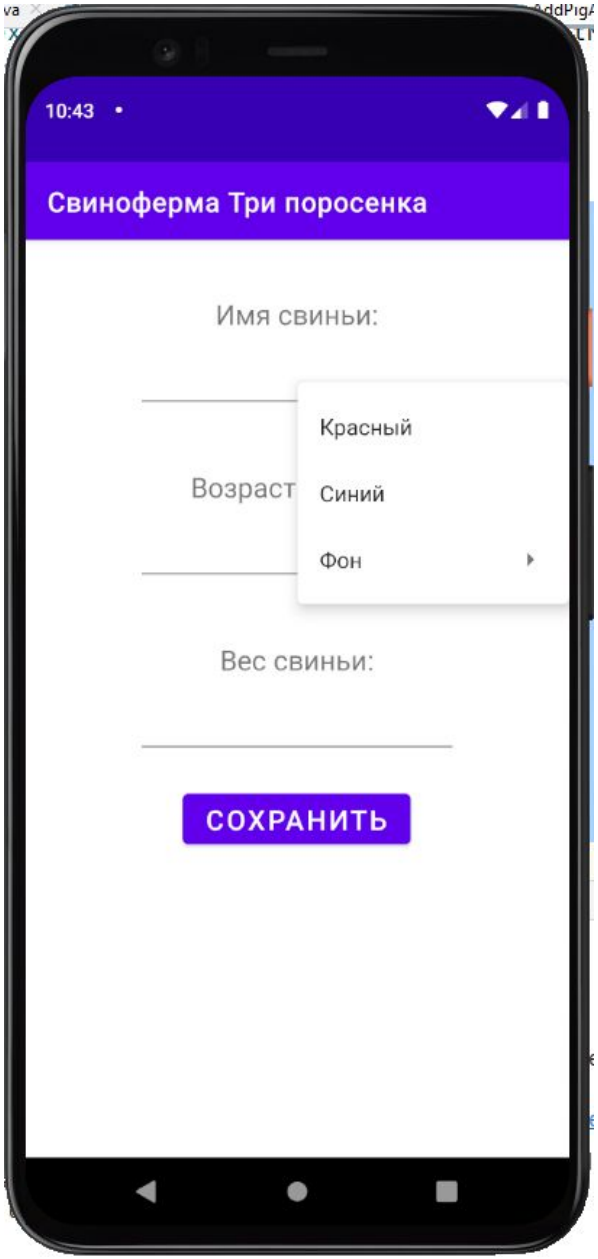
- Реализовать метод **onCreateContextMenu()** в активности

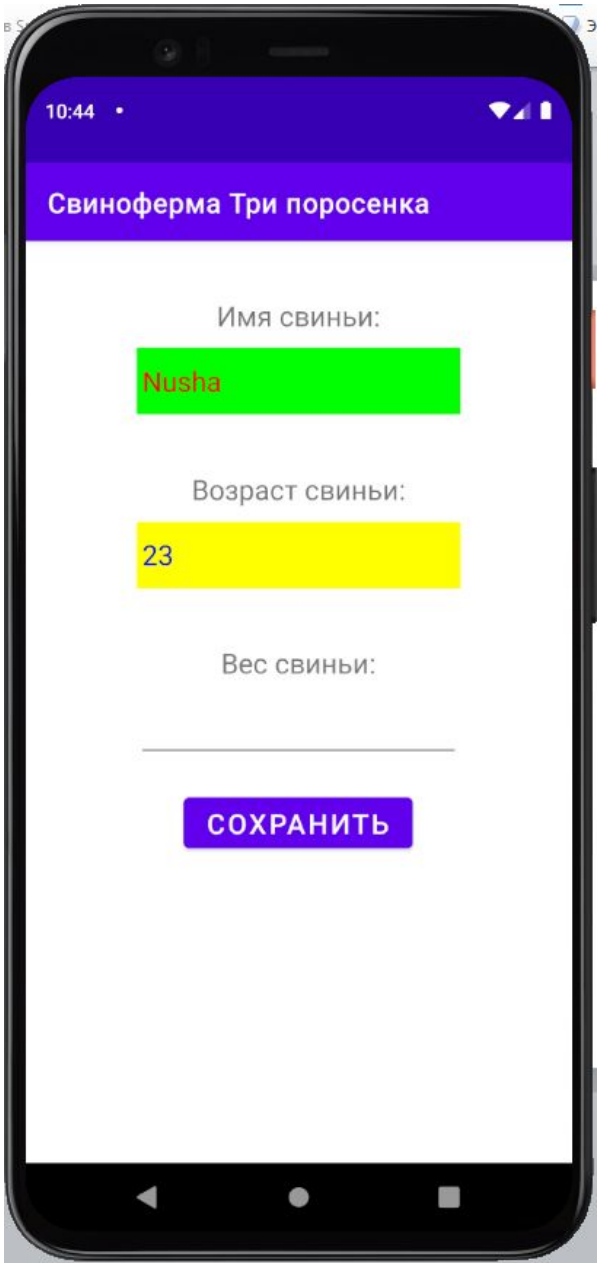
```
public void onCreateContextMenu(ContextMenu menu, View v,

ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    getMenuInflater().inflate(
            R.menu.layout_context_menu,menu);
}
```

Когда зарегистрированное представление примет событие длительного нажатия, система вызовет метод onCreateContextMenu(). Именно здесь определяются пункты меню. Делается это обычно путем загрузки ресурса меню.

- Реализовать метод **onContextItemSelected()**. Когда пользователь выбирает пункт меню, система вызывает этот метод, с тем чтобы можно было выполнить соответствующее действие:

```
public boolean onContextItemSelected(MenuItem item) {

    switch (item.getItemId()) {
        case R.id.menuItem_red:
            selectedEditText.setTextColor(Color.RED);
            break;
        case R.id.menuItem_blue:
            selectedEditText.setTextColor(Color.BLUE);
            break;
        case R.id.menuItem_yellow:
            selectedEditText.setBackgroundColor(Color.YELLOW);
            break;
        case R.id.menuItem_green:
            selectedEditText.setBackgroundColor(Color.GREEN);
            break;
    }
    return true;
}
```


10:43
Свиноферма Три поросенка
Имя свиньи:
Возраст
Красный
Синий
Фон
Вес свиньи:
СОХРАНИТЬ


10:44
Свиноферма Три поросенка
Имя свиньи:
Nusha
Возраст свиньи:
23
Вес свиньи:
СОХРАНИТЬ

Реализация режима контекстных действий.

Он представляет собой системную реализацию класса **ActionMode**. Когда пользователь использует этот режим, выбирая элемент, вверху экрана открывается строка контекстных действий, содержащая действия, которые пользователь может выполнить с выбранными в данный момент элементами.

Режим контекстных действий отключается, а строка контекстных действий исчезает, когда пользователь снимет выделение со всех элементов, нажмет кнопку Назад или выберет действие Готово, расположенное с левой стороны строки