

ЗАНЯТИЕ 2.18

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

Тема: Создание клиент-серверного приложения с использованием паттерна MVP.

Упражнение №1

Создайте новое Android-приложение для телефона с использованием шаблона Empty Activity (или Empty View Activity, в зависимости от версии Android Studio).

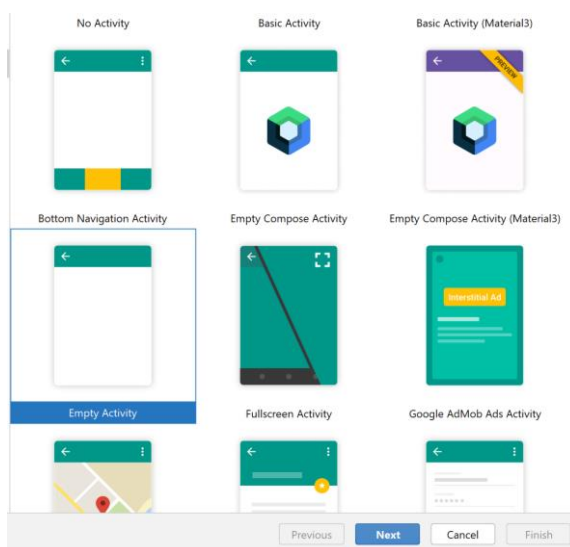


Рисунок 1

Установите минимальную версию API 21. Язык разработки – Java (не Kotlin). Имя приложения – StudentApp.

Рисунок 2

Дождитесь создания приложения и корректного обновления плагина Gradle. Перейдите в файл макета (в режим Code) **activity_main.xml**. Скопируйте код макета activity в свой проект. Изучите, какие виджеты

содержатся в макете! Значения текста и цвета указываются из файлов ресурсов strings.xml и colors.xml соответственно:

```
<?xml version="1.0" encoding="utf-8"?>
<android.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <EditText
        android:id="@+id/etLastName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/student_lastname"
        android:padding="8dp"
        android:textSize="24sp"
        android:layout_margin="32dp"
        app:layout_constraintTop_toTopOf="parent" />
    <EditText
        android:id="@+id/etGroup"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="32dp"
        android:hint="@string/group"
        android:padding="8dp"
        android:textSize="24sp"
        app:layout_constraintTop_toBottomOf="@+id/etLastName" />
    <Button
        android:id="@+id/btnInfo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/info"
        android:layout_margin="32dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/etGroup" />
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_margin="32dp"
        app:layout_constraintTop_toBottomOf="@+id/btnInfo">
        <RelativeLayout
            android:id="@+id/rlContent"
            android:visibility="gone"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
            <ImageView
                android:id="@+id/ivAvatar"
                android:layout_width="150dp"
                android:layout_height="150dp"
                android:layout_centerHorizontal="true" />
            <TextView
                android:id="@+id/tvInfo"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerHorizontal="true"
                android:layout_marginTop="16dp"
                android:textSize="20sp"
                android:layout_below="@id/ivAvatar"/>
        </RelativeLayout>
    </RelativeLayout>
</android.constraintlayout.widget.ConstraintLayout>
```

```

</RelativeLayout>
<ProgressBar
    android:id="@+id/progressBar"
    android:layout_width="75dp"
    android:layout_height="75dp"
    android:visibility="gone"
    android:layout_centerInParent="true"/>
</RelativeLayout>
</android.constraintlayout.widget.ConstraintLayout>

```

```

<resources>
    <string name="app_name">StudentApp</string>
    <string name="student_lastname">Student lastname</string>
    <string name="group">Group</string>
    <string name="info">Info</string>
</resources>

```

Рисунок 3 – Ссылки на строки в файле strings.xml

В макете содержатся два EditText для считывания фамилии и номера группы студента, а также кнопка, по нажатию на которую с сервера будет загружаться информация о студенте по введенным данным. Информация о студенте (аватар и характеристика) после загрузки будет отображаться в виджетах ImageView (id/ivAvatar) и TextView (id/tvInfo) соответственно. Оба этих виджета находятся в отдельном контейнере RelativeLayout (id/rlContent). Виджет ProgressBar будет использоваться для отображения хода загрузки.

Макет в итоге должен выглядеть примерно следующим образом:

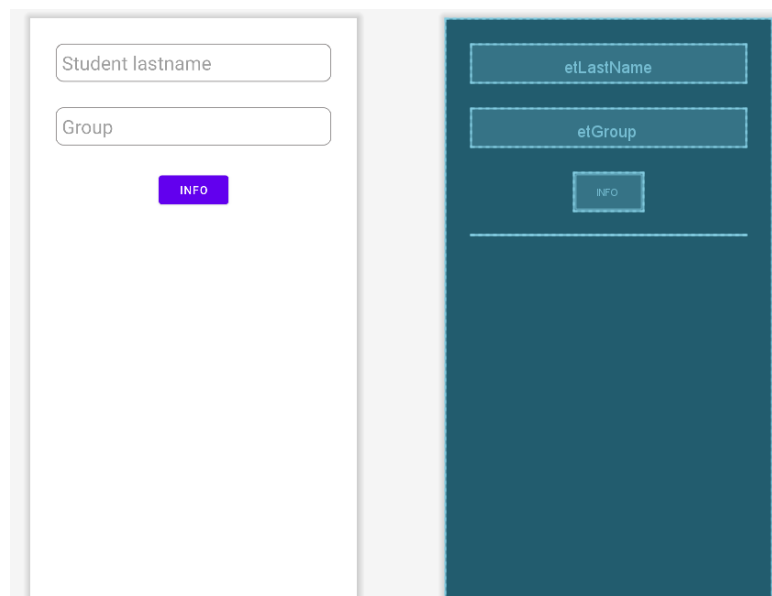


Рисунок 4

В приложении есть главный класс – Application. Это стандартный класс из Android SDK, и он используется неявно (вы с ним не работаете напрямую). Но можно создать собственный класс Application, с которого будет стартовать приложение. Как правило, он используется для хранения глобальных данных

и состояний, доступных во всем приложении. Класс `Application` наследуется от `Context`, что дает ему доступ к ресурсам и сервисам `Android`.

Создайте новый класс. Назовите его `StdApp` (или как-то иначе). Сделайте данный класс наследником класса `Application`. Переопределите в нем метод `onCreate` родительского класса (используйте средства генерации кода). Также создайте публичную тестовую константу (имя может быть любое, в данном примере – `LOG_TAG`), которая будет хранить тэг логирования, и к которой можно будет обращаться из любой точки приложения.

```
public class StdApp extends Application {

    public static final String LOG_TAG = "student_app_tag";

    @Override
    public void onCreate() {
        super.onCreate();
        Log.i(LOG_TAG, msg: "StudentApplication created!");
    }

}
```

Рисунок 5

Для того, чтобы работа приложения начиналась именно с этого наследника класса `Application`, необходимо добавить запись об этом в файле манифеста:

```
<application
    android:allowBackup="true"
    android:name=".StdApp"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
```

Рисунок 6

В данном приложении будут использоваться различные внешние библиотеки. Подключение внешних зависимостей происходит с помощью конфигурации файла `build.gradle`.

Добавьте в файл `build.gradle (Module)` поддержку библиотеки `Data Binding`, которая позволяет эффективнее и быстрее работать с `View`-компонентами и избавиться от большого количества лишнего кода.

```

buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
    }
}
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
}

buildFeatures {
    viewBinding true
}

```

Рисунок 7

После любых изменений файла `gradle`, необходимо синхронизировать зависимости с проектом (кнопка Sync Now):

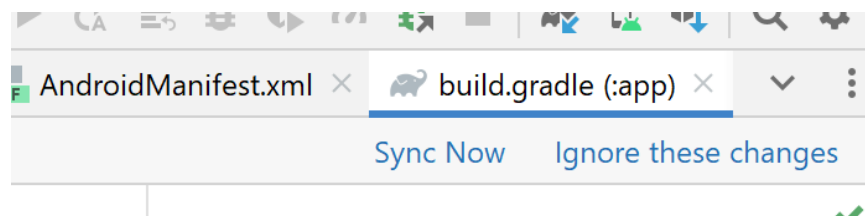


Рисунок 8

Перейдите в класс вашей activity и создайте переменную класса, которая будет являться ссылкой на объект `binding` («байндинг»). Именно через эту переменную будет осуществляться доступ ко всем виджетам макета по заданному Id. Следует обратить внимание, что тип этой переменной **зависит от имени файла** используемого вами макета. Так, для файла макета с именем `activity_main.xml` тип будет `ActivityMainBinding` и т.п. Инициализация переменной происходит с помощью `LayoutInflater` (напоминание: `LayoutInflater` – это класс, который умеет из содержимого `layout`-файла создать View-элемент). Теперь метод `onCreate` вашей activity должен выглядеть следующим образом:

```

private ActivityMainBinding binding;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding = ActivityMainBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());
}

```

Рисунок 9

Зарегистрируйте слушателя для кнопки «Info». В качестве слушателя используется текущий класс activity (обратите внимание, как теперь происходит доступ к виджетам с помощью Data Binding Library):

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
        binding.btnInfo.setOnClickListener(this);
    }

    @Override
    public void onClick(View view) {
        //...
    }
}
```

Рисунок 10

Так как разрабатываемое приложение будет «общаться» с сервером, то необходимо понимать какие запросы этот сервер принимает и какой формат данных возвращает в ответ. В этом и заключается смысл API. Сервер, которому будет отправлять запросы приложение умеет обрабатывать HTTP-запросы. Какой запрос нужно отправить серверу, чтобы получить информацию о студенте?

Сервер с удовольствием вернет ответ на такой GET-запрос:

URL запроса

Адрес сервера

Параметры запроса

Рисунок 11

В ответ на такой запрос сервер вернет данные в формате JSON. В случае, если информация о студенте найдена, то будет что-то подобное:

```
{
    "result_code": 1,
    "message_type": "data",
    "message_text": "Bla-bla-bla info about student",
    "avatar": "something url"
}
```

Рисунок 12

В случае, если информация о студенте не найдена или при обработке данных произошла ошибка:

```
{
    "result_code": 0,
    "message_type": "error",
    "message_text": "info about error"
}
```

Рисунок 13

Для отправки запросов на сервер и обработки ответа в данном проекте будет использоваться библиотека **Retrofit2**. Библиотека позволяет быстро получать и разбирать JSON (или другие структурированные данные) с помощью специальных конверторов, которые необходимо указывать вручную. В качестве HTTP-клиента использует внутри себя OkHttp.

Добавьте в файл build.gradle (Module) две зависимости в поле dependencies: библиотеку Retrofit2 и конвертор JSON (ведь как было показано ранее сервер возвращает данные именно в таком формате). После нажмите «Sync Now».

```
dependencies {
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
```

Рисунок 14

Для использования библиотеки Retrofit2 необходимо выполнить ряд действий. На первом этапе нужно подготовить классы моделей, которые соответствуют JSON-данным, которые возвращает сервер. Это, так называемый, POJO. POJO — «старый добрый Java-объект», простой Java-объект, не унаследованный от какого-то специфического объекта и не реализующий никаких служебных интерфейсов сверх тех, которые нужны для бизнес-модели. Такой класс можно сгенерировать онлайн на основе JSON-данных.

Перейдите на сайт <https://www.jsonschema2pojo.org/>, заполните поля имени пакета (туда нужно вписать имя пакета вашего приложения) и имени генерируемого класса (в данном примере выбрано имя StudentInfoResponse). Вставьте в поле для ввода текст JSON, который возвращает сервер на наш запрос:

```
{
    "result_code": 1,
    "message_type": "data",
    "message_text": "Bla-bla-bla info about student",
    "avatar": "something url"
}
```

Настройки на сайте: Source type – JSON, Annotation style – Gson, Include getters and setters.

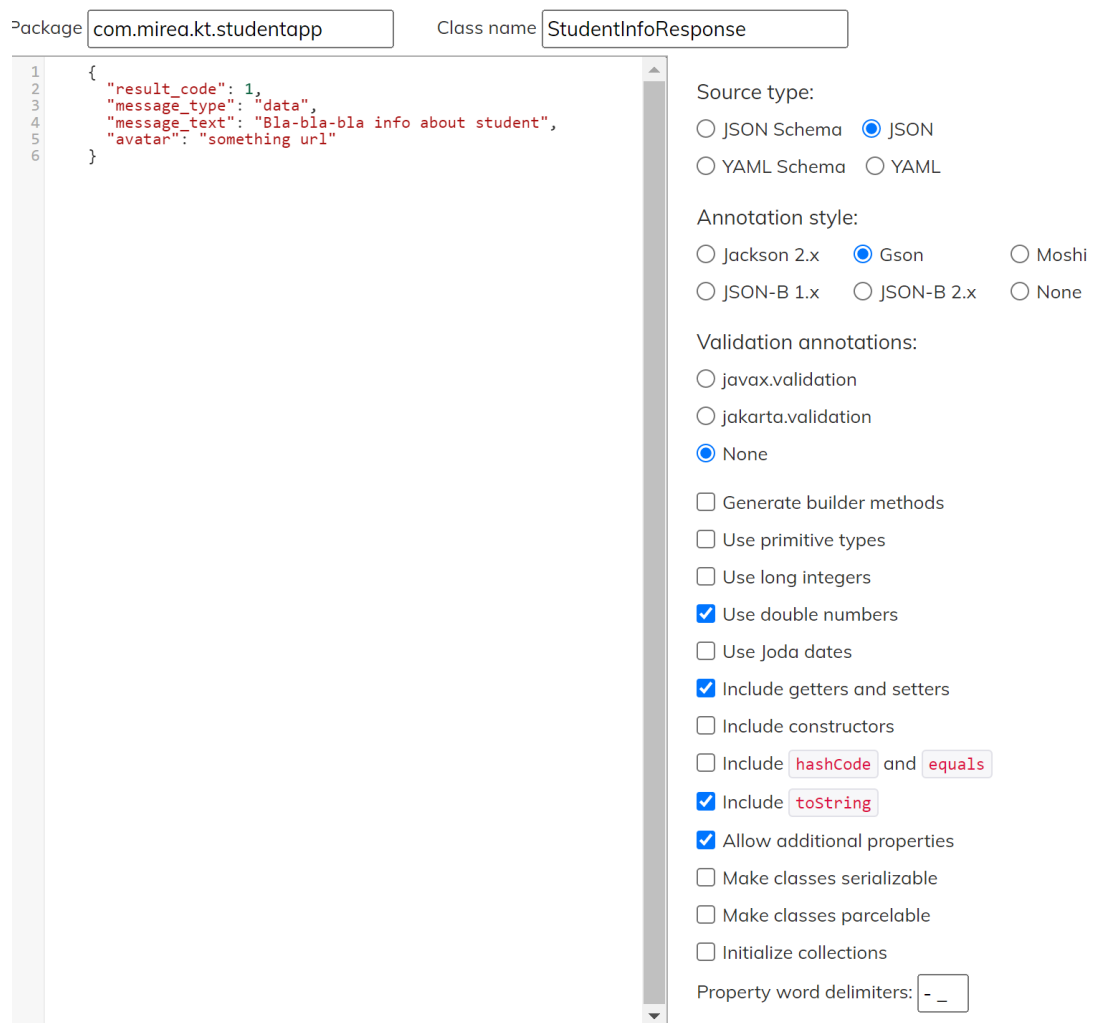


Рисунок 15

Нажмите кнопку «Preview» и скопируйте сгенерированный код класса. В вашем проекте создайте класс с таким же именем и перенесите туда скопированный код.

```

package com.mirea.kt.studentapp;

import ...

public class StudentInfoResponse {
    @SerializedName("result_code")
    @Expose
    private Integer resultCode;
    @SerializedName("message_type")
    @Expose
    private String messageType;
    @SerializedName("message_text")
    @Expose
    private String messageText;
    @SerializedName("avatar")
    @Expose
    private String avatar;

    public Integer getResultCode() { return resultCode; }
    public void setResultCode(Integer resultCode) { this.resultCode = resultCode; }
}

```

Рисунок 16

Добавьте требуемые классы в `import`.

Доп. информация для интересующихся, как описать POST-запрос (не обязательно к выполнению):

```
@POST("/materials/practical/info.php")
@FormUrlEncoded
Call<StudentInfoResponse> getStudentInfoAll(@Field("name")String n, @Field("group")String gr);
```

Рисунок 18

На следующем этапе нужно подготовить объект интерфейса Api, чтобы можно было выполнять подготовленные на предыдущем шаге запросы. Перейдите в класс-наследник Application, который был создан ранее и добавьте код инициализации объекта Retrofit и Api. Ссылку на Api нужно сделать статической, а также реализовать для нее геттер – для доступа к ней из любой точки приложения.

```
public class StdApp extends Application {

    public static final String LOG_TAG = "student_app_tag";
    private static Api api;

    @Override
    public void onCreate() {
        super.onCreate();
        Log.i(LOG_TAG, msg: "StudentApplication created!");
        Retrofit rt = new Retrofit.Builder()
            .baseUrl("https://android-for-students.ru") // к какому серверу будут запросы
            .addConverterFactory(GsonConverterFactory.create()) // какой конвертор использовать
            .build();
        api = rt.create(Api.class);
    }

    public static Api getServerApi() {
        return api;
    }
}
```

Рисунок 19

Изучите данный код, при необходимости задайте вопрос преподавателю.

Данный проект разрабатывается согласно «чистой архитектуры» с использованием паттерна MVP – Model View Presenter. Presenter – это объект, который управляет отображением данных через специальный интерфейс View. Presenter также обращается к слою данных (Repository) за получением этих самых данных и занимается обработкой жизненного цикла. View – это интерфейс, содержащий методы для работы с UI, который реализуется в Activity или Fragment. Model – обычные классы сущностей данных предметной области (в данном случае некоторый StudentInfo). С ними работает слой Repository.

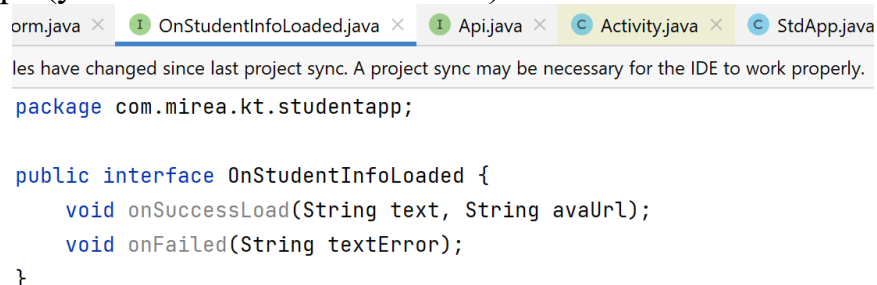
Далее будет представлена одна из возможных реализаций этого паттерна.

Создайте класс MainContract. Почему Main? Потому что в нем будут описаны интерфейсы паттерна MVP для экрана с **MainActivity** (то есть названия выбираются исключительно для удобства и понимания). В созданном классе опишите интерфейсы для слоев: View, Presenter, Repository. Каждый слой выполняет свои задачи, для каждого будут соответствующие методы. Скопируйте код класса в свой проект.

```
public class MainContract {
    interface MainView{
        void showLoading();
        void hideLoading();
        void showMessage(int resId);
        void showMessage(String message);
        void showError(int resId);
        void showError(String errorMessage);
        void showStudentInfo(String text, String avaUrl);
    }
    interface MainPresenter{
        void onViewCreated();
        void onClickGetInfoButton(String name, String group);
        void onViewDestroyed();
    }
    interface MainRepository{
        void loadStudentInfo(String n, String g, OnStudentInfoLoaded sil);
    }
}
```

Обратите внимание, что интерфейс для слоя View содержит методы только для работы с UI. Методы Presenter отвечают за логику приложения. Метод слоя данных (Repository) отвечает за получение данных с сервера.

Для обратной связи Presenter-Repository можно использовать Listener (слушатель) в виде интерфейса. Создайте в проекте приложения новый интерфейс. Название может быть любое, в примере – OnStudentInfoLoaded. Добавьте в него пару методов, через которые в Presenter будет возвращаться ответ сервера (успешный или ошибочный):



```
orm.java × OnStudentInfoLoaded.java × Api.java × Activity.java × StdApp.java
les have changed since last project sync. A project sync may be necessary for the IDE to work properly.
package com.mirea.kt.studentapp;

public interface OnStudentInfoLoaded {
    void onSuccessLoad(String text, String avaUrl);
    void onFailed(String textError);
}
```

Рисунок 20

Описание слоев паттерна MVP завершено. Теперь можно использовать эти интерфейсы. Предлагается начать с уровня данных. Создайте в проекте новый класс – MainRepository. Реализуйте в данном классе ранее разработанный интерфейс.

```

public class MainRepository implements MainContract.MainRepository{

    @Override
    public void loadStudentInfo(String name, String group, OnStudentInfoLoaded sil) {

    }

}

```

Рисунок 21

В методе loadStudentInfo необходимо реализовать запрос к серверу и получение информации о студенте с помощью библиотеки Retrofit2:

```

public class MainRepository implements MainContract.MainRepository{

    @Override
    public void loadStudentInfo(String name, String group, OnStudentInfoLoaded sil) {
        StdApp.getServerApi().getStudentInfo(name,group).enqueue(new );
    }
}

```

1 Callback<StudentInfoResponse>{...} (retrofit2)
 2 MainRepository com.mirea.kt.studentapp
 3 MainPresenter com.mirea.kt.studentapp
 4 Api com.mirea.kt.studentapp

Рисунок 22

Используется ранее подготовленная ссылка на Api (из класса Application) и соответствующий метод (реализующий тот самый GET-запрос). Для выполнения запроса используется асинхронный метод enqueue.

```

@Override
public void loadStudentInfo(String name, String group, OnStudentInfoLoaded sil) {
    StdApp.getServerApi().getStudentInfo(name,group).enqueue(new Callback<StudentInfoResponse>() {
        @Override
        public void onResponse(Call<StudentInfoResponse> call, Response<StudentInfoResponse> response) {
            StudentInfoResponse studentInfoResponse = response.body();
            if(studentInfoResponse != null){
                int rezCode = studentInfoResponse.getResultCode();
                if(rezCode == 1){
                    sil.onSuccessLoad(studentInfoResponse.getMessageText(),studentInfoResponse.getAvatar());
                }else if(rezCode == 0){
                    sil.onFailed(studentInfoResponse.getMessageText());
                }else {
                    sil.onFailed( textError: "Unknown result code");
                }
            }else{
                sil.onFailed( textError: "Empty server response");
            }
        }
        @Override
        public void onFailure(Call<StudentInfoResponse> call, Throwable t) {
            sil.onFailed(t.getMessage());
        }
    });
}

```

Рисунок 23

Обратите внимание, что при обработке ответа от сервера используется уже не JSON, а объект класса модели – StudentInfoResponse. Библиотека Retrofit2 автоматически преобразовала JSON-строку тела ответа с помощью конвертора в объект этого класса.

С помощью ссылки на слушатель OnStudentInfoLoaded значения возвращаются в то место, откуда был вызван метод loadStudentInfo. А вызван он будет из Presenter. Подготовьте его.

Создайте в проекте новый класс – MainPresenter. Реализуйте в данном классе ранее разработанный интерфейс. Обратите внимание, что в Presenter содержатся ссылки и на View и на Repository. Через них будут вызываться методы этих слоев. Значение View будет передаваться в конструкторе.

```
public class MainPresenter implements MainContract.MainPresenter{

    private MainContract.MainView view;
    private MainRepository repository;

    public MainPresenter(MainContract.MainView view) {
        this.view = view;
    }

    @Override
    public void onViewCreated() {
        // какая-то логика, которая выполняется при создании View
    }

    @Override
    public void onClickGetInfoButton(String name, String group) {

    }

    @Override
    public void onViewDestroyed() {
        // какая-то логика, которая выполняется при уничтожении View
    }
}
```

Рисунок 24

Скопируйте код метода onClickGetInfoButton:

```
public void onClickGetInfoButton(String name, String group) {
    if(this.repository == null){
        this.repository = new MainRepository();
    }
    if(!name.isEmpty() && !group.isEmpty()){
        view.showLoading(); //запускаем крутиться Progressbar
        repository.loadStudentInfo(name, group, new OnStudentInfoLoaded()
{
    @Override
    public void onSuccessLoad(String text, String avaUrl) {
        view.hideLoading(); //остановка Progressbar
        view.showStudentInfo(text,avaUrl); // показать инфо о
студенте
    }
    @Override
    public void onFailed(String textError) {
        view.showError(textError); // показать ошибку
    }
});
    }else{
        view.showError(R.string.error_empty_fields); // показать ошибку
    }
}
```

Этот метод будет вызываться из View при нажатии на кнопку получения информации о студенте. В данном методе происходят все проверки (то есть логика) введенных значений, а также передача этих значений дальше в Repository. С помощью экземпляра анонимного класса типа OnStudentInfoLoaded в Presenter возвращается информация о студенте или информация о возникшей ошибке. Текст данных о студенте или текст ошибки передаются обратно в UI-слой для отображения. Также вызываются методы отображения и скрывания ползунка ProgressBar. То есть перед загрузкой – показать ProgressBar, после окончания загрузки – скрыть.

Теперь пора реализовать UI-слой. В данном случае UI-уровень работает в MainActivity, значит разработанный ранее интерфейс MainContract.MainView должен реализовываться именно этим классом:

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener, MainContract.MainView {
```

Рисунок 25

Переопределите методы этого интерфейса (это можно сделать автоматически средствами генерации кода). Добавьте в методы код:

```
@Override
public void showLoading() {
    binding.progressBar.setVisibility(View.VISIBLE);
}

@Override
public void hideLoading() {
    binding.progressBar.setVisibility(View.GONE);
}

@Override
public void showMessage(int resId) {
    Toast.makeText(context: this, resId, Toast.LENGTH_LONG).show();
}

@Override
public void showError(String errorMessage) {
    Toast.makeText(context: this, errorMessage, Toast.LENGTH_LONG).show();
}

@Override
public void showError(int resId) {
    Toast.makeText(context: this, resId, Toast.LENGTH_LONG).show();
}

@Override
public void showStudentInfo(String text, String aVaUrL) {
    // здесь будет код отображения инфы о студенте
}
```

Рисунок 26

```
<string name="error_empty_fields">Необходимо заполнить поля ввода!</string>
```

Рисунок 27 – Хранение строки в файле strings.xml

Для отображения картинки (аватарки) в ImageView будет использоваться внешняя библиотека Picasso. Добавьте зависимость в build.gradle:

```
implementation 'com.squareup.picasso:picasso:2.8'
```

После этого воспользуйтесь этой библиотекой для загрузки картинки по ее URL-адресу:

```
@Override
public void showStudentInfo(String text, String avaUrl) {
    // здесь будет код отображения инфы о студенте
    binding.rlContent.setVisibility(View.VISIBLE);
    binding.tvInfo.setText(text);
    Picasso.get().load(avaUrl)
        .fit() // растянуть картинку по содержимому
        .into(binding.ivAvatar);
}
```

Рисунок 27

Создайте в View ссылку на Presenter и вызовите соответствующий метод Presenter-а по нажатию на кнопку «Info»:

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener, MainC
```

```
private ActivityMainBinding binding;
private MainContract.MainPresenter presenter;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding = ActivityMainBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());
    binding.btnInfo.setOnClickListener(this);
    presenter = new MainPresenter( view: this);
}

@Override
public void onClick(View view) {
    String lastName = binding.etLastName.getText().toString();
    String group = binding.etGroup.getText().toString();
    presenter.onClickGetInfoButton(lastName, group);
}
```

Рисунок 28

Добавьте в файл манифеста разрешения для работы с интернет:

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

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

Рисунок 29

Запустите приложение, введите свою фамилию на латинице, номер группы и нажмите на кнопку. **Убедитесь, что загруженные данные успешно отображаются.**

Дополнительная информация для интересующихся, как закруглить содержимое ImageView? Ведь обычно аватар отображается в круглом виде.

Это можно сделать с помощью библиотеки Picasso. Скопируйте в проект класс преобразования картинки в круг:

```
public class CircleTransform implements Transformation {

    boolean mCircleSeparator = false;

    public CircleTransform() {
    }

    public CircleTransform(boolean circleSeparator) {
        mCircleSeparator = circleSeparator;
    }

    @Override
    public Bitmap transform(Bitmap source) {
        int size = Math.min(source.getWidth(), source.getHeight());

        int x = (source.getWidth() - size) / 2;
        int y = (source.getHeight() - size) / 2;

        Bitmap squaredBitmap = Bitmap.createBitmap(source, x, y, size, size);

        if (squaredBitmap != source) {
            source.recycle();
        }

        Bitmap bitmap = Bitmap.createBitmap(size, size, source.getConfig());

        Canvas canvas = new Canvas(bitmap);
        BitmapShader shader = new BitmapShader(squaredBitmap,
        BitmapShader.TileMode.CLAMP, BitmapShader.TileMode.CLAMP);
        Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG | Paint.DITHER_FLAG |
        Paint.FILTER_BITMAP_FLAG);
        paint.setShader(shader);

        float r = size/2f;
        canvas.drawCircle(r, r, r, paint);
    }
}
```



```

//border:
Paint paintBorder = new Paint();
paintBorder.setStyle(Paint.Style.STROKE);
paintBorder.setColor(Color.argb(84,0,0,0));
paintBorder.setAntiAlias(true);
paintBorder.setStrokeWidth(1);
canvas.drawCircle(r, r, r-1, paintBorder);

if (mCircleSeparator) {
    Paint paintBorderSeparator = new Paint();
    paintBorderSeparator.setStyle(Paint.Style.STROKE);
    paintBorderSeparator.setColor(Color.parseColor("#ffffff"));
    paintBorderSeparator.setAntiAlias(true);
    paintBorderSeparator.setStrokeWidth(4);
    canvas.drawCircle(r, r, r+1, paintBorderSeparator);
}

squaredBitmap.recycle();
return bitmap;
}

@Override
public String key() {
    return "circle";
}
}

```

Теперь объект этого класса можно использовать при отображении картинки в ImageView:

```

@Override
public void showStudentInfo(String text, String avaUrl) {
    binding.rlContent.setVisibility(View.VISIBLE);
    binding.tvInfo.setText(text);
    Picasso.get().load(avaUrl)
        .fit()
        .transform(new CircleTransform())
        .into(binding.ivAvatar);
}

```

Рисунок 30