Data Binding

Data binding - это support library, которая позволяет связывать компоненты UI в макетах с источниками данных в приложении, используя декларативный формат, а не программно.

https://developer.android.com/topic/libraries/data-binding

Макеты часто определяются в действиях с кодом. Например:

```
TextView textView = findViewById(R.id.text1);
textView.setText(viewModel.getUserName());
```

При использовании библиотеки привязки можно назначить текст виджету в файле макета. Это устраняет необходимость в коде Java. Используется синтаксис @{} в выражении присваивания:

```
<TextView
    android:id="@+id/text1"
    android:text="@{viewmodel.userName}"
```

Для выполнения привязки необходимо:

1) В **build.gradle** файл модуля в секции android необходимо включить Data Binding:

```
android {
    compileSdkVersion 30
    buildToolsVersion "30.0.2"

    ...
    dataBinding {
        enabled = true
    }
}
```

2) *Определить объект данных*. Например, entity типа Student:

```
public class Student {

    public String firstName;
    public   String secondName;

    public Student(String firstName, String secondName) {
        this.firstName = firstName;
        this.secondName = secondName;
    }
}
```

3) *Измененить layout*. Язык выражений позволяет писать выражения, обрабатывающие события, отправляемые layout. Библиотека привязки данных автоматически генерирует классы, необходимые для привязки представлений в макете к объектам данных.

Но для этого файлы макета должны начинаться с корневого тега макета **layout**, за которым следуют элемент данных и корневой элемент представления:

```
<?xml version="1.0" encoding="utf-8"?>
<layout>

<androidx.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">

    <TextView
        android:id="@+id/fname"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="34sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.552"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.205" />

    <TextView
        android:id="@+id/sName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="116dp"
        android:textSize="34sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.549"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/fname" />

</androidx.constraintlayout.widget.ConstraintLayout>

</layout>
```

Корневым элементом теперь является <layout>, а ConstraintLayout сместился внутрь него. Data Binding по созданным xml-файлам генерирует специальные классы, которые и управляют всеми процессами передачи данных.

Для каждого файла макета создается класс привязки. По умолчанию имя класса - имя файла макета с добавлением суффикса Binding. Имя файла макета - *activity_main.xml*, поэтому соответствующий сгенерированный класс - **ActivityMainBinding**. Этот класс содержит все привязки от свойств макета

(например, пользовательские переменные) к представлениям макета и знает, как назначать значения для выражений привязки.

Рекомендуемый метод создания привязок:

```
public class MainActivity extends AppCompatActivity {

    private ActivityMainBinding activityMainBinding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);

        activityMainBinding = DataBindingUtil.setContentView(this,
R.layout.activity_main);
        activityMainBinding.setStudent(new Student("Anna","Petrova"));

    }
}
```

Метод *setStudent* был сгенерирован в классе биндинга, т.к. была описана переменная *student* в *layout* файле. Этим методом мы передаем биндингу объект Student. Биндинг возьмет значения student.firstName и student.secondName и поместит их (методом setText) в соответствующие TextView.

Если нужно использовать элементы привязки данных внутри адаптера Fragment, ListView или RecyclerView, можно использовать метод *inflate*() классов привязок или класса **DataBindingUtil**:

```
ListItemBinding binding = ListItemBinding.inflate(layoutInflater, viewGroup,
false);
// or
        ListItemBinding binding = DataBindingUtil.inflate(layoutInflater,
R.layout.list_item, viewGroup, false);
```

4) *сделать привязку в layout* на основе Expression language.

Язык выражений позволяет писать выражения, обрабатывающие события, отправляемые представлениями.

Внутри тега <*data*> можно объявлять переменные с помощью тегов <*variable*>, в качестве атрибутов указываем имя переменной и ее тип. Далее эти переменные можно использовать в разметке и получать из них данные:

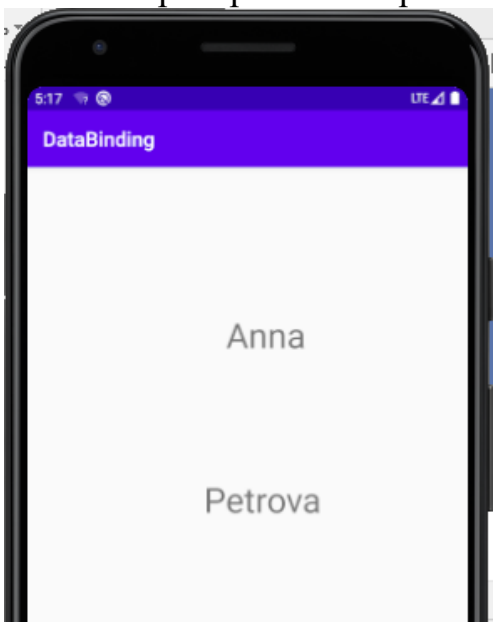
```
<layout >
    <data>
        <variable
            name="student"
            type="by.bstu.patsei.databinding.Student" />
    </data>
...
```

Здесь пользовательская переменная *student* описывает свойство, которое может использоваться в этом макете. Выражения в макете записываются в свойствах атрибутов с использованием синтаксиса «@{}»:

https://developer.android.com/topic/libraries/data-binding/expressions

```
<TextView
    android:id="@+id/fname"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="34sp"
    android:text="@{student.firstName}"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.552"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.205" />

<TextView
    android:id="@+id/sName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="116dp"
    android:textSize="34sp"
    android:text="@{student.secondName}"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.549"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/fname" />
```

Запускоем приложение и проверяем что привязка работает



Однако надо учитывать что, если в коде будет изменяться объект *Student*, то данные на экране меняться не будут. Они считались один раз и далее не отслеживаются (при такой реализации).

Чтобы экран получил новые данные, надо снова передать биндингу измененный объект или вызывать метод *invalidateAll*().

Data Binding для событий

С помощью баиндинга мы можем назначать обработчики на события View. Добавим две кнопки *Ok* и *Cancel*.

```
<Button
    android:id="@+id/okButton"
    android:layout_width="0dp"
    android:layout_weight="1"
    android:text="Ok"
    android:layout_height="wrap_content"/>

<Button
    android:id="@+id/cancelButton"
    android:layout_width="0dp"
    android:layout_weight="1"
    android:text="Cancel"
    android:layout_height="wrap_content"/>
```

Внутири класса активности создаем внутреннйи (inner) класс с обработчиками и контекстом (понадобиться контекст активности):

```
public class MainActivityButtonHandlers{
        Context context;

        public MainActivityButtonHandlers(Context context) {
            this.context = context;
        }
        public void onOkClicked(View view) {
            Toast.makeText(context, "OK", Toast.LENGTH_LONG).show();
        }

        public void onCancelClicked(View view) {
            Toast.makeText(context, "Cancel", Toast.LENGTH_LONG).show();
        }

    }
//    public void onOkClicked(View view) {
//        Toast.makeText(this, "OK", Toast.LENGTH_LONG).show();
//    }
//
//    public void onCancelClicked(View view) {
//        Toast.makeText(this, "Cancel", Toast.LENGTH_LONG).show();
//    }
```

Прописываем обработчик, как *variable* в layout.

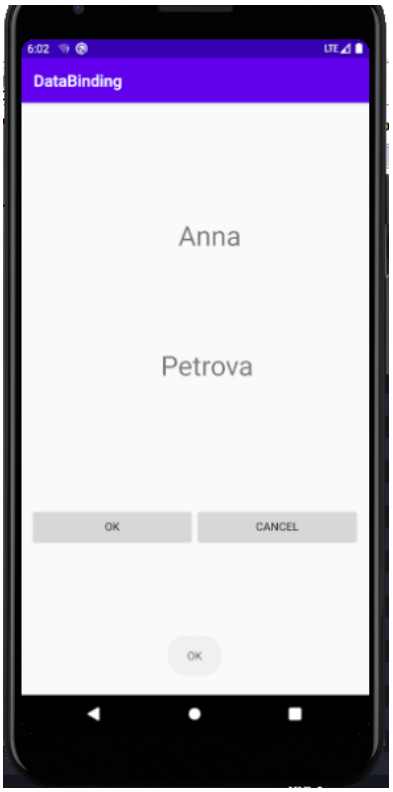
```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">
    <data>
        <variable
            name="student"
            type="by.bstu.patsei.databinding.Student" />

        <variable
            name="buttonHandler"

type="by.bstu.patsei.databinding.MainActivity.MainActivityButtonHandlers" />
    </data>
```

Осталось создать объект класса с обработчикам в активности и передать его в биндинг:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //setContentView(R.layout.activity_main);


    activityMainBinding = DataBindingUtil.setContentView(this,
R.layout.activity_main);
    activityMainBinding.setStudent(new Student("Anna","Petrova"));

    mainActivityButtonHandlers = new MainActivityButtonHandlers(this);
    activityMainBinding.setButtonHandler(mainActivityButtonHandlers);
}
```

В *onClick* кнопки ссылаемся на его методы:

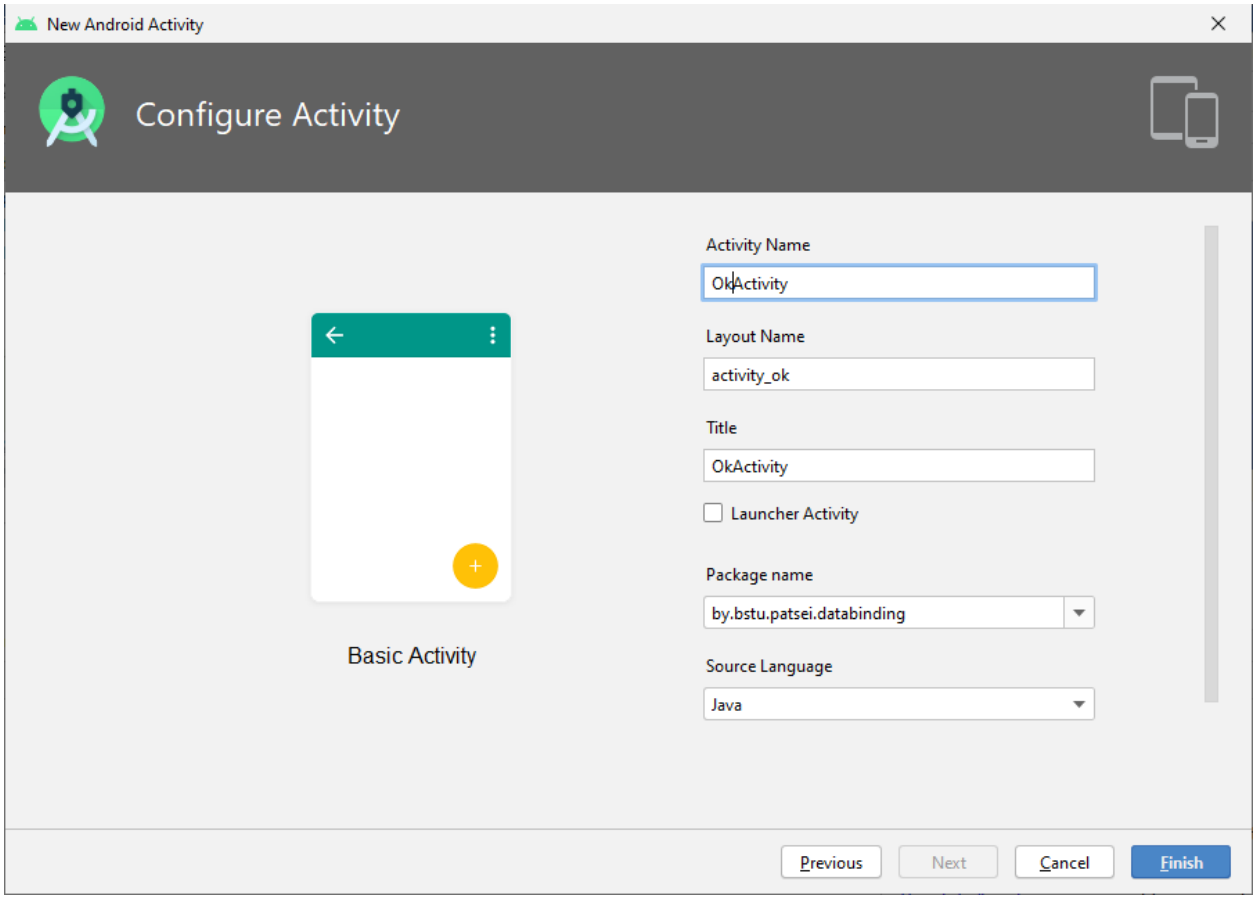
```
<Button
    android:id="@+id/okButton"
    android:layout_width="0dp"
    android:layout_weight="1"
    android:text="Ok"
    android:onClick="@{buttonHandler::onOkClicked}"
    android:layout_height="wrap_content"/>

<Button
    android:id="@+id/cancelButton"
    android:layout_width="0dp"
    android:layout_weight="1"
    android:text="Cancel"

android:onClick="@{buttonHandler::onCancelClicked}"
    android:layout_height="wrap_content"/>
```

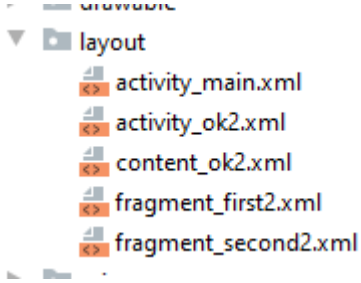


Использование include

Часто есть несколько файлов разметки и они соединяются с помощью тега ***include***. С помощью *include* можно использовать один *layout* файл внутри другого *layout* файла с binding и передавать переменные.

Добавим еще одну активность, которую назовем *OkActivity*





В *activity*_ok2.xml уже включен *content*_ok2.xml

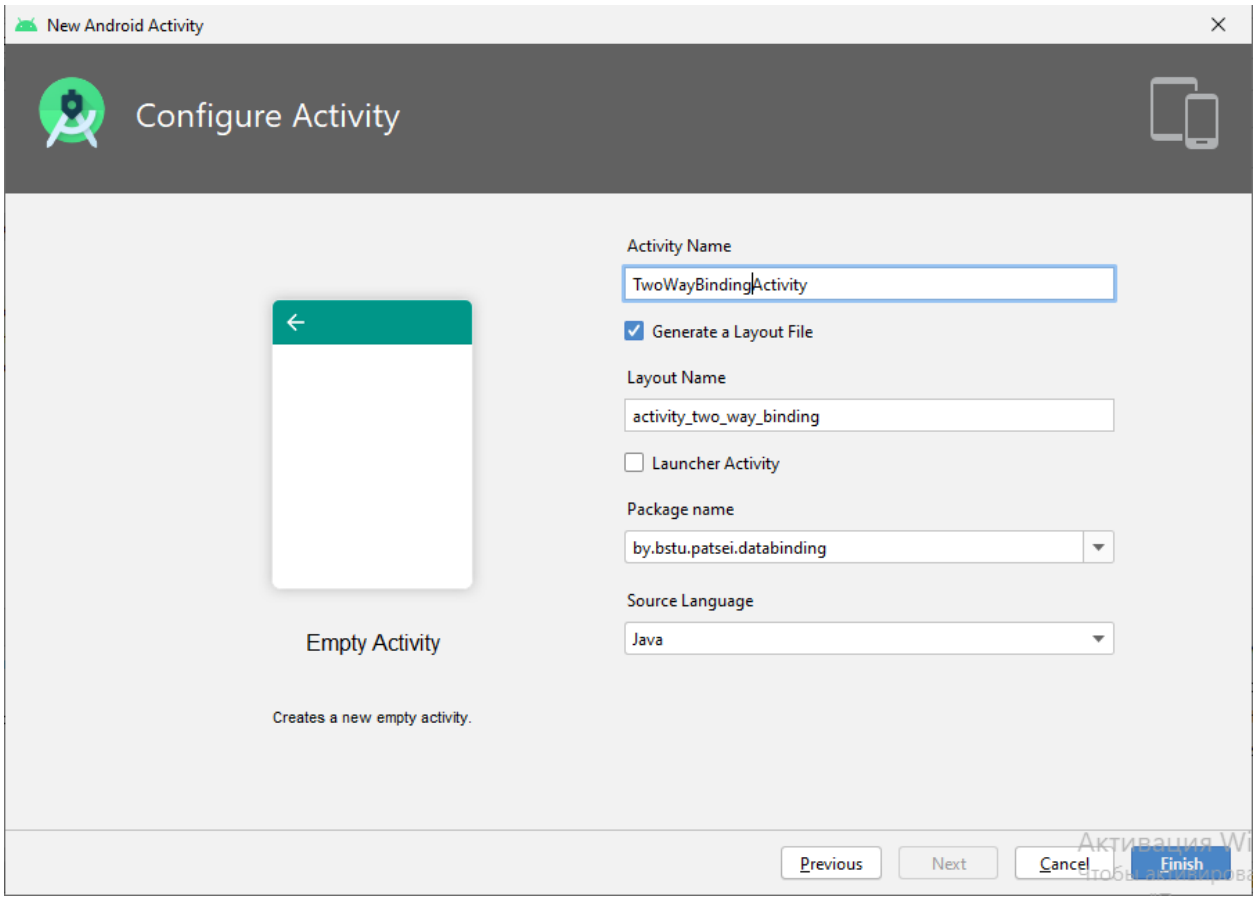
```
<androidx.coordinatorlayout.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".OkActivity">

    <com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

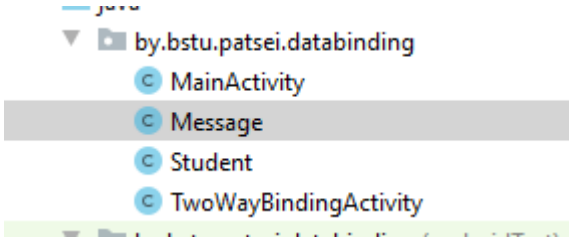
```
        android:theme="@style/AppTheme.AppBarOverlay">

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />

    </com.google.android.material.appbar.AppBarLayout>

    <include layout="@layout/content_ok2" />

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fab"
```

Two –way Data Binding

Баиндинг может работать в обе стороны. Т.е. он будет не только передавать данные во *View*, но и получать их оттуда.

Создадим еще одну активность и назовем *TwoWayBindingActivity*





И создадим entity класс *Message*

```
public class Message {

    private  String messageSender;
    private  String messageText;

}
```

Меняем разметку *activity_two_way_binding.xml*. Добавляем перемнную *message* и TextView. Делаем привязку.

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

    <data>
        <variable
            name="message"
            type="by.bstu.patsei.databinding.Message" />
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".TwoWayBindingActivity">


        <TextView
            android:id="@+id/messageText"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="34sp"
            android:text="@{message.messageText}"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintHorizontal_bias="0.552"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintVertical_bias="0.205" />

    </androidx.constraintlayout.widget.ConstraintLayout>

</layout>
```

В активности тоже деалем привязку.

```
public class TwoWayBindingActivity extends AppCompatActivity {

    private ActivityTwoWayBindingBinding activityTwoWayBindingBinding;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    //    setContentView(R.layout.activity_two_way_binding);

        activityTwoWayBindingBinding = DataBindingUtil.setContentView(this,
R.layout.activity_two_way_binding);
        activityTwoWayBindingBinding.setMessage(getCurrentMessage());
    }

    private Message getCurrentMessage(){
        return new Message("Fit","Hello");
    }
}
```

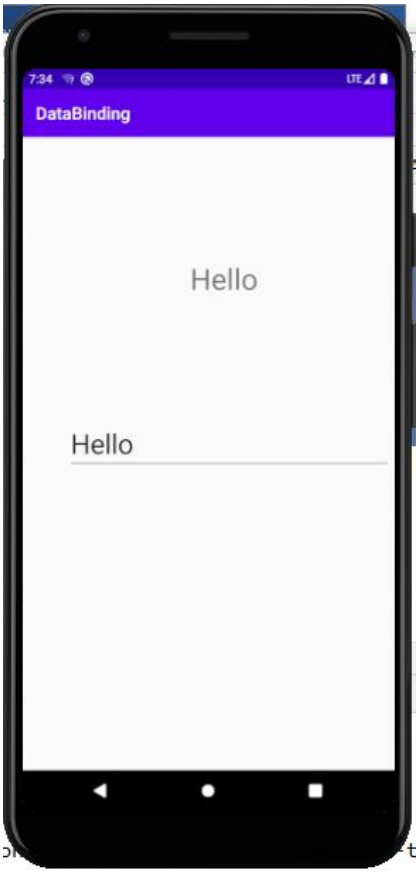
И добавлем переход из первой активности во вторую активность при нажатии кнопки Ok

```
public class MainActivityButtonHandlers{
    Context context;

    public MainActivityButtonHandlers(Context context) {
        this.context = context;
    }
    public void onOkClicked(View view) {
        Toast.makeText(context, "OK", Toast.LENGTH_LONG).show();
        startActivity(new Intent(MainActivity.this, TwoWayBindingActivity.class));
    }

    public void onCancelClicked(View view) {
        Toast.makeText(context, "Cancel", Toast.LENGTH_LONG).show();
    }

}
```

Запускаем.Проверяем переход – работает.

Добавим в эту активность еще *EditText* и тоже с привязкой

```
<?xml version="1.0" encoding="utf-8"?>
<layout 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">

    <data>
        <variable
            name="message"
            type="by.bstu.patsei.databinding.Message" />
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".TwoWayBindingActivity">


        <TextView
            android:id="@+id/messageText"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="34sp"
            android:text="@{message.messageText}"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintHorizontal_bias="0.552"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintVertical_bias="0.205" />

        <EditText
            android:id="@+id/editText"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="125dp"
            android:layout_marginTop="331dp"
            android:layout_marginEnd="77dp"
            android:layout_marginBottom="355dp"
            android:ems="10"
            android:textSize="32sp"
            android:inputType="textPersonName"
            android:text="@{message.messageText}"

app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

    </androidx.constraintlayout.widget.ConstraintLayout>

</layout>
```



Пока она не двухсторонняя. Для того чтобы сдаелать two-way

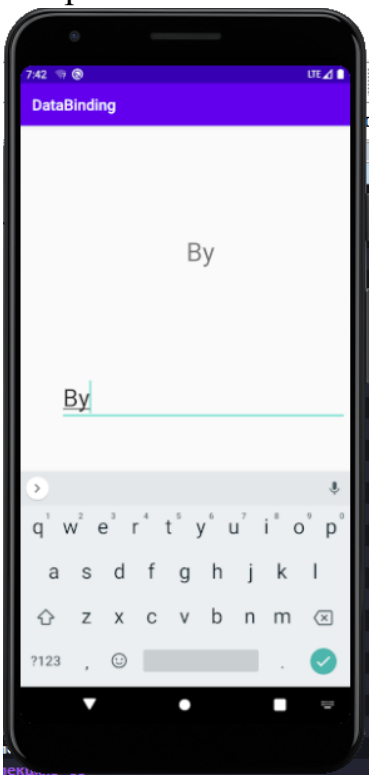
1) Поставить знак = после @

```
<EditText
    android:id="@+id/editText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="125dp"
```

```
        android:layout_marginTop="331dp"
        android:layout_marginEnd="77dp"
        android:layout_marginBottom="355dp"
        android:ems="10"
        android:textSize="32sp"
        android:inputType="textPersonName"
        android:text="@={message.messageText}"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

2) Нужно изменить класть Message

Чтобы указать, что значение может меняться и что эти изменения нужно отслеживать, используется аннотация **@Bindable**, которая применяется к методу для получения значения какого-то поля. Если после этого выполнить сборку проекта, то мы увидим, что в сгенерированном классе **BR** (этот класс похож на класс R – класс всех ресурсов, только служит для идентификаторов всех свойств) появится поле. Это поле используется для уведомления о том, что связанное поле класса было изменено. Дальше нужно унаследовать класс *Message* от **BaseObservable**. И теперь мы можем использовать метод **notifyPropertyChanged** и передать ему идентификатор изменившегося свойства.



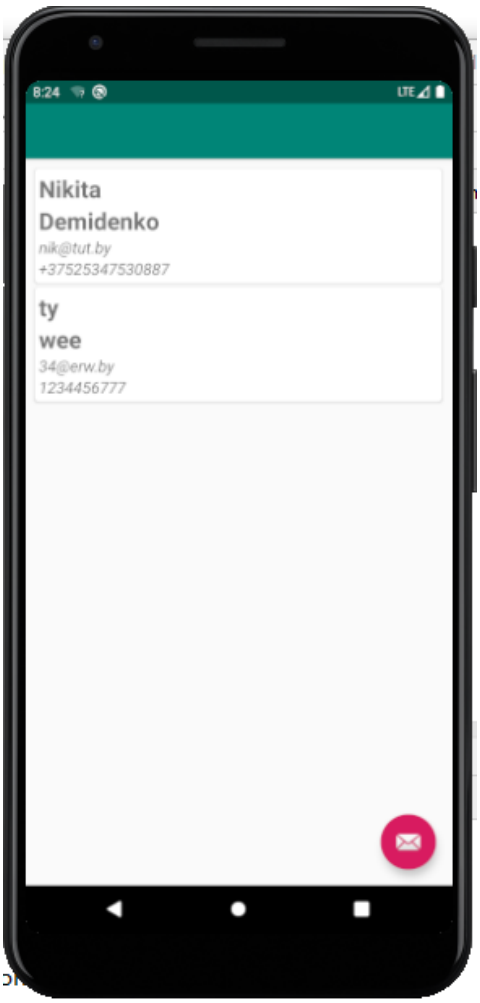

```
public class Message extends BaseObservable
{
    public  String messageSender;
    public  String messageText;
    @Bindable
    public String getMessageSender() {
        return messageSender;
    }
    @Bindable
    public String getMessageText() {
        return messageText;
    }
    public void setMessageSender(String
messageSender) {
        this.messageSender = messageSender;

notifyPropertyChanged(BR.messageSender);
    }
    public void setMessageText(String
messageText) {
        this.messageText = messageText;
        notifyPropertyChanged(BR.messageText);
    }
    public Message() {    }

    public Message(String messageSender, String messageText) {
        this.messageSender = messageSender;
        this.messageText = messageText;
    }
}
```

Итак, автобиндинг для Java объекта делается наследованием ***BaseObservable***. Поля будут отслеживаться биндингом если пометить get-методы аннотацией **@Bindable**, а в set-методах вызывать **notifyPropertyChanged** метод, который и будет уведомлять биндинг об изменениях значения поля.

Есть и другие способы для реализации связывания, например, можно использовать классы ObservableInt,…, ObservableField<Type> и другие.

Data Binding для RecycleView

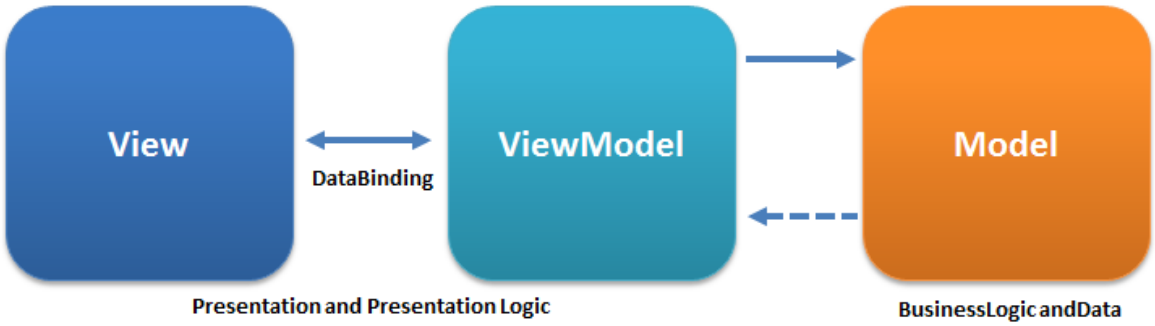
Пример, как использовать биндинг для элементов списка RecylerView посмотрите в примере.
https://github.com/PatseiBSTU/DataBindingContactList.git



MVVM

Ключевой особенностью паттерна MVVM является то, что ни один компонент (Model, View, ViewModel) не знает о другом явно. Эти компоненты взаимодействуют между собой за счет механизма связывания данных (Bindings).

Изменение данных во *ViewModel* автоматически меняет данные, отображаемые во *View*. Аналогично, любое событие или изменение данных во *View* (нажатие на кнопку, ввода текста и другое) изменяет данные во *ViewModel*. Это позволяет не хранить явные ссылки на *View* во *ViewModel* и наоборот, а также держать эти компоненты очень слабо связными, что удобно при тестировании и не только.



Представление

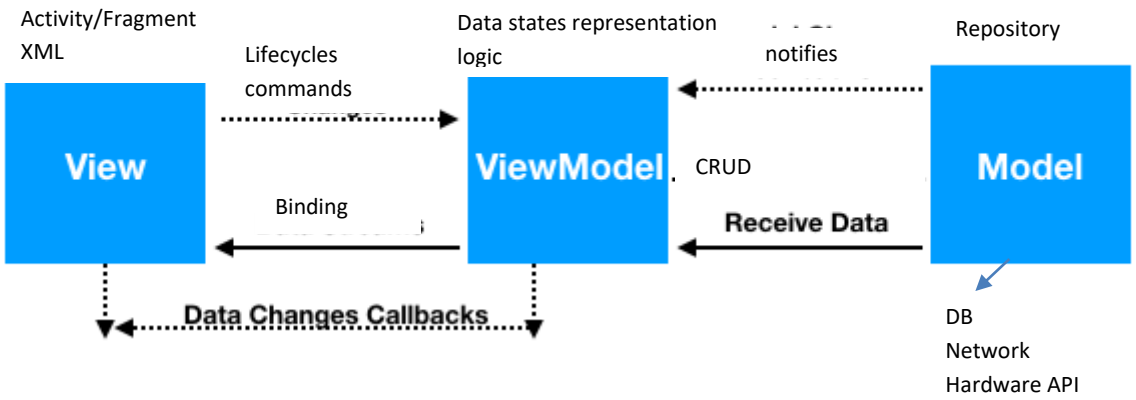
Содержит структурное определение того, что пользователи получат на экранах. Вы можете поместить сюда статическое и динамическое содержимое (анимацию и смену состояний). Для нашего случая в представлении может быть активность или фрагмент. **View == Activity** (т.е. XML + Java-class).

Модель представления

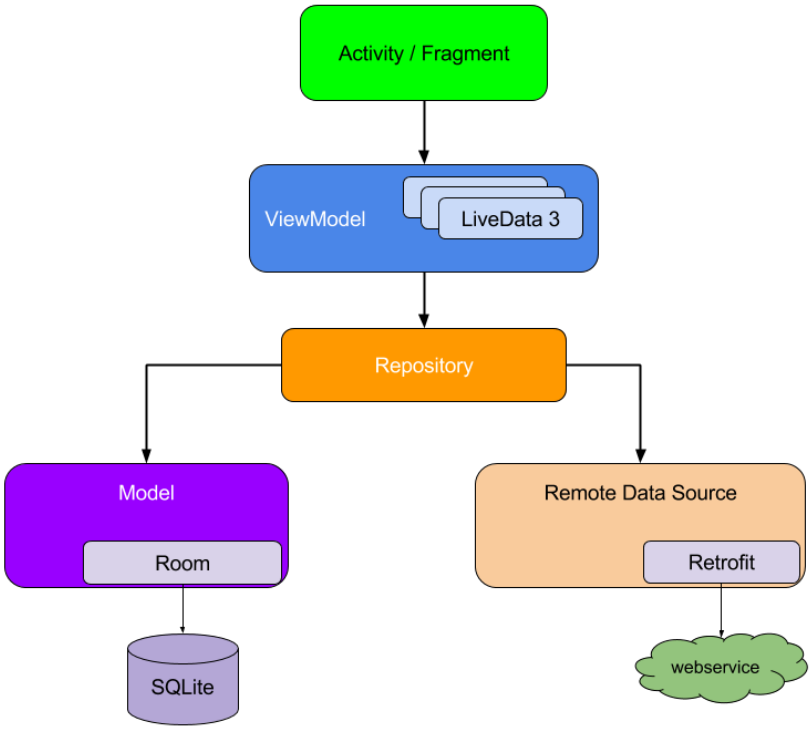
Этот компонент связывает модель и представление. Отвечает за управление ссылками данных и возможных конверсий. Здесь появляется биндинг. В Android мы не беспокоимся об этом, потому что можно напрямую использовать класс **AndroidViewModel** или **ViewModel.**

Модель

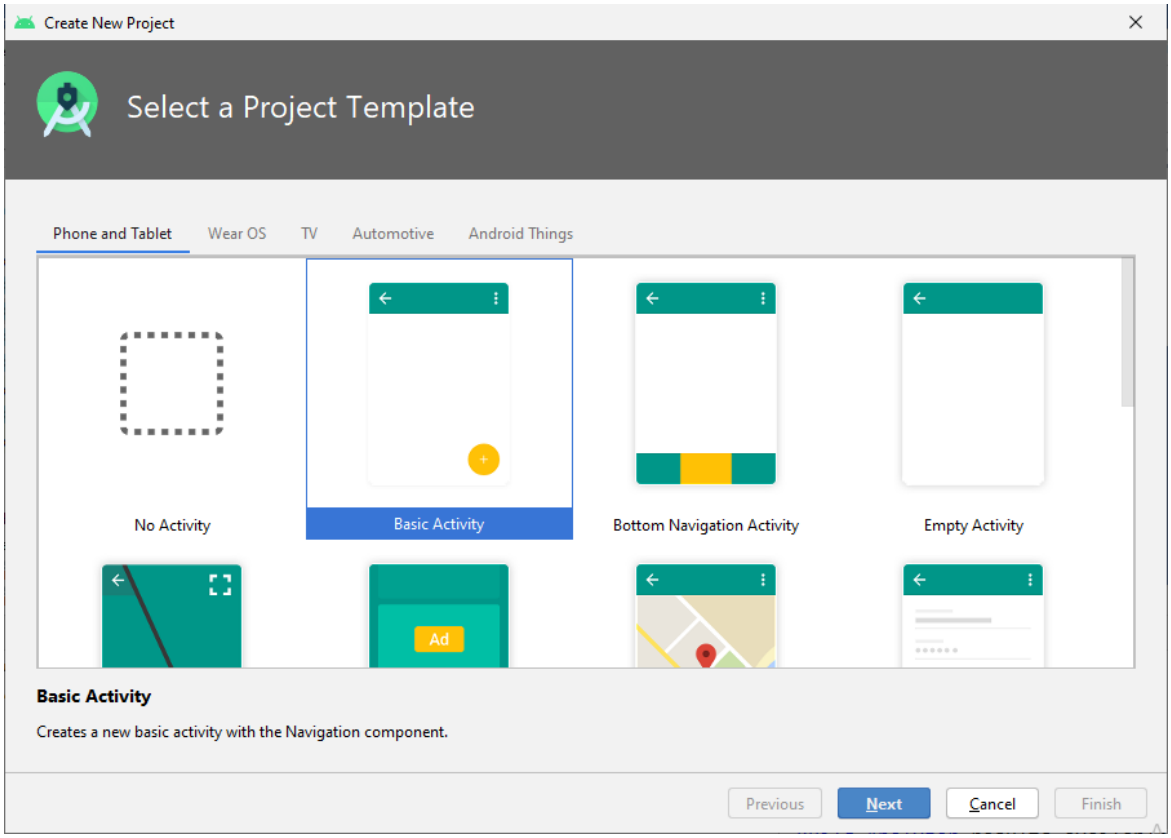
Это уровень бизнес-данных. В Android, согласно “чистой” архитектуре, модель может содержать базу данных, репозиторий и класс бизнес-логики.

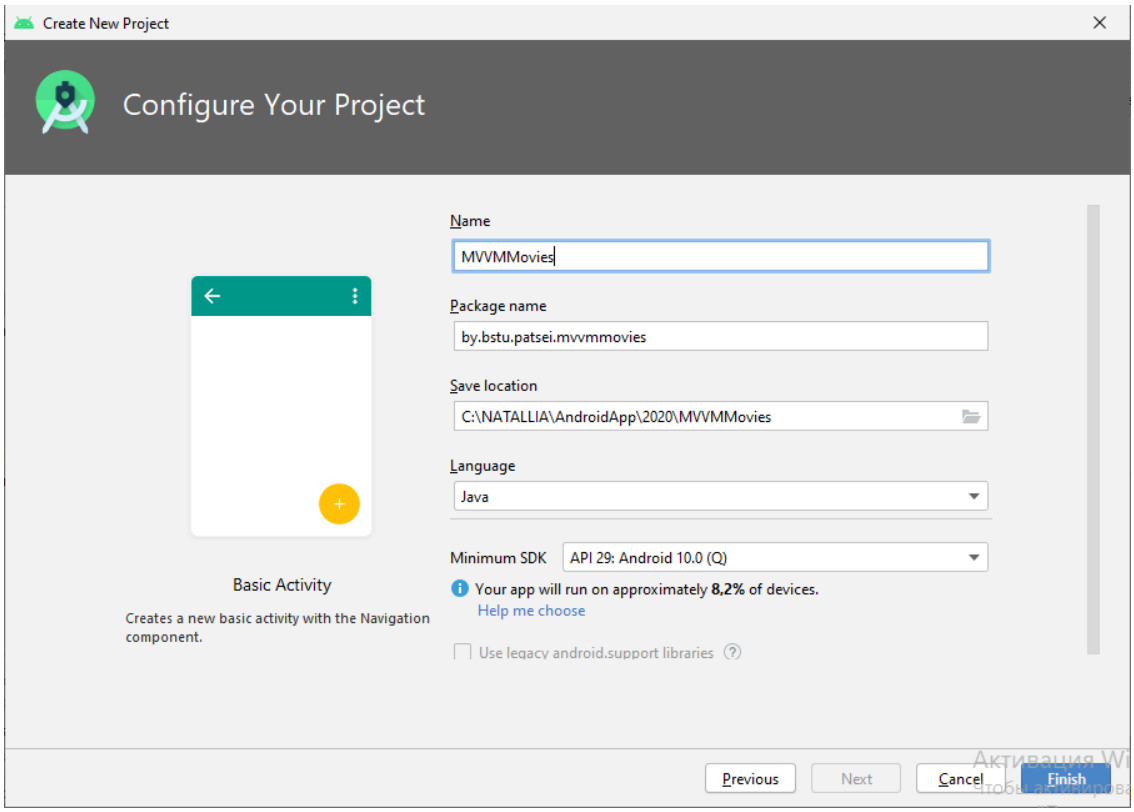


Чтобы понять, как функционирует паттерн MVVM, напишем небольшое приложение, в котором будут все компоненты с картинки. Приложение будет сохранять данные локально для того, чтобы оно работало в режиме оффлайн.



Создадим новый проект MVVMMovies.





Подключаем все нужные зависимости , в том числе и databinding

```
def lifecycle_version = "2.1.0"
// ViewModel
implementation "androidx.lifecycle:lifecycle-viewmodel:$lifecycle_version"
// LiveData
implementation "androidx.lifecycle:lifecycle-livedata:$lifecycle_version"
// Annotation processor
annotationProcessor "androidx.lifecycle:lifecycle-compiler:$lifecycle_version"

def room_version = "2.2.3"
implementation "androidx.room:room-runtime:$room_version"
annotationProcessor "androidx.room:room-compiler:$room_version"

dataBinding {
    enabled = true
}
```

Создаем пакет **model**. У нас будет две таблицы *Genre* и *Movie*. Оба типа являются **BaseObservbale** и **Entitiy**. Сделаем возможность изменения значений в полях на основе data binding (рассматривали выше).

```
@Entity(tableName = "genres_table")
public class Genre extends BaseObservable {
    @PrimaryKey(autoGenerate = true)
    private int id;
    @ColumnInfo(name = "genre_name")
    private String genreName;
    @Ignore
    public Genre() {
    }
    public Genre(int id, String genreName) {
        this.id = id;
```

```
        this.genreName = genreName;
    }
    @Bindable
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
        notifyPropertyChanged(BR.id);
    }
    @Bindable
    public String getGenreName() {
        return genreName;
    }
    public void setGenreName(String genreName) {
        this.genreName = genreName;
        notifyPropertyChanged(BR.genreName);
    }
    @Override
    public String toString() {
        return this.genreName;
    }
}

@Entity(tableName = "movies_table", foreignKeys = @ForeignKey(entity = Genre.class,
parentColumns = "id", childColumns = "genre_id", onDelete = ForeignKey.CASCADE))
public class Movie extends BaseObservable {

    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "movie_id")
    private int movieId;
    @ColumnInfo(name = "movie_name")
    private String movieName;
    @ColumnInfo(name = "movie_description")
    private String movieDescription;
    @ColumnInfo(name = "genre_id")
    private int genreId;
    @Ignore
    public Movie() {
    }
    public Movie(int movieId, String movieName, String movieDescription, int genreId)
{
        this.movieId = movieId;
        this.movieName = movieName;
        this.movieDescription = movieDescription;
        this.genreId = genreId;
    }
    @Bindable
    public int getMovieId() {
        return movieId;
    }
    public void setMovieId(int movieId) {
        this.movieId = movieId;
        notifyPropertyChanged(BR.movieId);
    }
    @Bindable
    public String getMovieName() {
        return movieName;
    }
    public void setMovieName(String movieName) {
        this.movieName = movieName;
        notifyPropertyChanged(BR.movieName);
    }
```

```
    @Bindable
    public String getMovieDescription() {
        return movieDescription;
    }
    public void setMovieDescription(String movieDescription) {
        this.movieDescription = movieDescription;
        notifyPropertyChanged(BR.movieDescription);
    }
    @Bindable
    public int getGenreId() {
        return genreId;
    }
    public void setGenreId(int genreId) {
        this.genreId = genreId;
        notifyPropertyChanged(BR.genreId);
    }
}
```

Создадим Dao для каждой таблицы:

```
@Dao
public interface GenreDao {
    @Insert
    void insert(Genre genre);
    @Update
    void update(Genre genre);
    @Delete
    void delete(Genre genre);
    @Query("select * from genres_table")
    LiveData<List<Genre>> getAllGenres();

}
```

```
@Dao
public interface MovieDao {
    @Insert
    void insert(Movie movie);
    @Update
    void update(Movie movie);
    @Delete
    void delete(Movie movie);
    @Query("select * from movies_table")
    LiveData<List<Movie>> getAllMovies();
    @Query("select * from movies_table where genre_id==:genreId")
    LiveData<List<Movie>> getGenreMovies(int genreId);

}
```

Создадим класс *MoviesDatabase* с двумя таблицами. И сделаем singlton для создания и миграции. Сделаем inner классы **RoomDatabase.Callback** и **InitialDataAsyncTask** они будут вызываться один раз при создании.

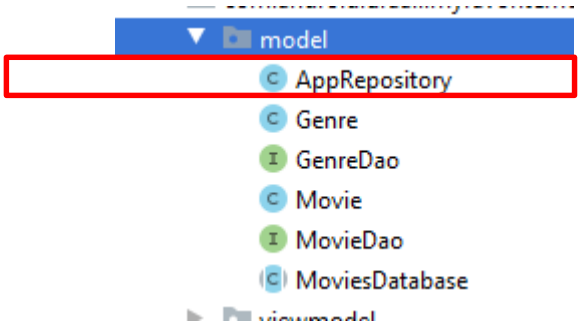
```
@Database(entities = {Genre.class, Movie.class}, version = 1)
public abstract class MoviesDatabase extends RoomDatabase {

    private static MoviesDatabase instance;
    public abstract GenreDao getGenreDao();
```

```
    public abstract MovieDao getMovieDao();

    public static synchronized MoviesDatabase getInstance(Context context) {
        if (instance == null) {
            instance = Room.databaseBuilder(context.getApplicationContext(),
                    MoviesDatabase.class, "moviesDB")
                    .fallbackToDestructiveMigration()
                    .addCallback(callback)
                    .build();
        }
        return instance;
    }

    private static RoomDatabase.Callback callback = new RoomDatabase.Callback(){
        @Override
        public void onCreate(@NonNull SupportSQLiteDatabase db) {
            super.onCreate(db);
            new InitialDataAsyncTask(instance).execute();
        }
    };

    private static class InitialDataAsyncTask extends AsyncTask<Void, Void, Void> {
        private GenreDao genreDao;
        private MovieDao movieDao;
        public InitialDataAsyncTask(MoviesDatabase database) {
            genreDao = database.getGenreDao();
            movieDao = database.getMovieDao();
        }

        @Override
        protected Void doInBackground(Void... voids) {

            Genre comedyGenre = new Genre();
            comedyGenre.setGenreName("Comedy");

            Genre romanceGenre = new Genre();
            romanceGenre.setGenreName("Romance");

            Genre dramaGenre = new Genre();
            dramaGenre.setGenreName("Drama");

            genreDao.insert(comedyGenre);
            genreDao.insert(romanceGenre);
            genreDao.insert(dramaGenre);

            Movie movie1 = new Movie();
            movie1.setMovieName("Bad Boys for Life");
            movie1.setMovieDescription("The Bad Boys Mike Lowrey and Marcus Burnett
are back together for one last ride in the highly anticipated Bad Boys for Life.");
            movie1.setGenreId(1);

            Movie movie2 = new Movie();
            movie2.setMovieName("Parasite");
            movie2.setMovieDescription("All unemployed, Ki-taek and his family take
peculiar interest in the wealthy and glamorous Parks, as they ingratiate themselves
into their lives and get entangled in an unexpected incident.");
            movie2.setGenreId(1);

            Movie movie3 = new Movie();
            movie3.setMovieName(" Once Upon a Time... in Hollywood");
            movie3.setMovieDescription("A faded television actor and his stunt double
strive to achieve fame and success in the film industry during the final years of
Hollywood's Golden Age in 1969 Los Angeles.");
```

```
            movie3.setGenreId(1);

            Movie movie4 = new Movie();
            movie4.setMovieName("You");
            movie4.setMovieDescription("A dangerously charming, intensely obsessive
young man goes to extreme measures to insert himself into the lives of those he is
transfixed by.");
            movie4.setGenreId(2);

            Movie movie5 = new Movie();
            movie5.setMovieName("Little Women");
            movie5.setMovieDescription("Jo March reflects back and forth on her life,
telling the beloved story of the March sisters - four young women each determined to
live life on their own terms.");
            movie5.setGenreId(2);

            Movie movie6 = new Movie();
            movie6.setMovieName("Vikings");
            movie6.setMovieDescription("Vikings transports us to the brutal and
mysterious world of Ragnar Lothbrok, a Viking warrior and farmer who yearns to
explore - and raid - the distant shores across the ocean.");
            movie6.setGenreId(2);

            Movie movie7 = new Movie();
            movie7.setMovieName("1917");
            movie7.setMovieDescription("Two young British soldiers during the First
World War are given an impossible mission: deliver a message deep in enemy territory
that will stop 1,600 men, and one of the soldiers' brothers, from walking straight
into a deadly trap.");
            movie7.setGenreId(3);

            Movie movie8 = new Movie();
            movie8.setMovieName("The Witcher");
            movie8.setMovieDescription("Geralt of Rivia, a solitary monster hunter,
struggles to find his place in a world where people often prove more wicked than
beasts.");
            movie8.setGenreId(3);

            Movie movie9 = new Movie();
            movie9.setMovieName("The Outsider");
            movie9.setMovieDescription("Investigators are confounded over an
unspeakable crime that's been committed.");
            movie9.setGenreId(3);

            movieDao.insert(movie1);
            movieDao.insert(movie2);
            movieDao.insert(movie3);
            movieDao.insert(movie4);
            movieDao.insert(movie5);
            movieDao.insert(movie6);
            movieDao.insert(movie7);
            movieDao.insert(movie8);
            movieDao.insert(movie9);

            return null;
        }
    }
}
```

Следующий этап создание репозитория. В том же пакете **model**. Репозиторий будет содержать Dao для каждой таблицы, которые

иинициализиуются в конструкторе и два ***LiveData*** со со списками фильмов и жанров. Раместим там методы получения, обновления жанров и фильмов. Сделаем возможность загрузки, обновления и удаления в асинхронном режиме, поэтому используем AsynTask.




```
public class AppRepository {

    private GenreDao genreDao;
    private MovieDao movieDao;

    private LiveData<List<Genre>> genres;
    private LiveData<List<Movie>> movies;

    public AppRepository(Application application) {
        MoviesDatabase database = MoviesDatabase.getInstance(application);
        genreDao = database.getGenreDao();
        movieDao = database.getMovieDao();
    }

    public LiveData<List<Genre>> getGenres() {
        return genreDao.getAllGenres();
    }

    public LiveData<List<Movie>> getGenreMovies(int genreId) {
        return movieDao.getGenreMovies(genreId);
    }

    public void insertGenre(Genre genre) {
        new InsertGenreAsyncTask(genreDao).execute(genre);
    }

    public void insertMovie(Movie movie) {
        new InsertMovieAsyncTask(movieDao).execute(movie);
    }

    private static class InsertGenreAsyncTask extends AsyncTask<Genre, Void, Void> {
        private GenreDao genreDao;
        public InsertGenreAsyncTask(GenreDao genreDao) {
            this.genreDao = genreDao;
        }

        @Override
        protected Void doInBackground(Genre... genres) {
            genreDao.insert(genres[0]);
            return null;
        }
    }
```

```
private static class InsertMovieAsyncTask extends AsyncTask<Movie, Void, Void> {
    private MovieDao movieDao;
    public InsertMovieAsyncTask(MovieDao movieDao) {
        this.movieDao = movieDao;
    }

    @Override
    protected Void doInBackground(Movie... movies) {
        movieDao.insert(movies[0]);
        return null;
    }
}

public void updateGenre(Genre genre) {
    new UpdateGenreAsyncTask(genreDao).execute(genre);
}

public void updateMovie(Movie movie) {
    new UpdateMovieAsyncTask(movieDao).execute(movie);
}

private static class UpdateGenreAsyncTask extends AsyncTask<Genre, Void, Void> {
    private GenreDao genreDao;
    public UpdateGenreAsyncTask(GenreDao genreDao) {
        this.genreDao = genreDao;
    }

    @Override
    protected Void doInBackground(Genre... genres) {
        genreDao.update(genres[0]);
        return null;
    }
}

private static class UpdateMovieAsyncTask extends AsyncTask<Movie, Void, Void> {
    private MovieDao movieDao;
    public UpdateMovieAsyncTask(MovieDao movieDao) {
        this.movieDao = movieDao;
    }

    @Override
    protected Void doInBackground(Movie... movies) {
        movieDao.update(movies[0]);
        return null;
    }
}

public void deleteGenre(Genre genre) {
    new DeleteGenreAsyncTask(genreDao).execute(genre);
}

public void deleteMovie(Movie movie) {
    new DeleteMovieAsyncTask(movieDao).execute(movie);
}

private static class DeleteGenreAsyncTask extends AsyncTask<Genre, Void, Void> {
    private GenreDao genreDao;
    public DeleteGenreAsyncTask(GenreDao genreDao) {
        this.genreDao = genreDao;
    }

    @Override
    protected Void doInBackground(Genre... genres) {
```

```
            genreDao.delete(genres[0]);
            return null;
        }
    }

    private static class DeleteMovieAsyncTask extends AsyncTask<Movie, Void, Void> {
        private MovieDao movieDao;
        public DeleteMovieAsyncTask(MovieDao movieDao) {
            this.movieDao = movieDao;
        }

        @Override
        protected Void doInBackground(Movie... movies) {
            movieDao.delete(movies[0]);
            return null;
        }
    }
}
```

Создадим ***ViewModel*** для активностей для отображения фильмов и для обновления и добавления фильма. Создаем пакет *viewmodel*. Если наследоваться от ***AndroidViewModel***, то можно будет использовать context. ***ViewModel*** работает с репозиторием:

```
public class MainActivityViewModel extends AndroidViewModel {

    AppRepository appRepository;
    private LiveData<List<Genre>> genres;
    private LiveData<List<Movie>> genreMovies;

    public MainActivityViewModel(@NonNull Application application) {
        super(application);
        appRepository = new AppRepository(application);
    }

    public LiveData<List<Genre>> getGenres() {
        genres = appRepository.getGenres();
        return genres;
    }

    public LiveData<List<Movie>> getGenreMovies(int genreId) {
       genreMovies = appRepository.getGenreMovies(genreId);
        return genreMovies;
    }

    public void addNewMovie(Movie movie) {
        appRepository.insertMovie(movie);
    }

    public void updateMovie(Movie movie) {
        appRepository.updateMovie(movie);
    }

    public void deleteMovie(Movie movie) {
        appRepository.deleteMovie(movie);
    }
}
```

Класс *ViewModel* создан для того, чтобы хранить и управлять данными, связанными с UI относительно жизненного цикла. Он позволяет данным пережить изменения конфигурации (например, повороты экрана).

Viewmodel принимает репозиторий в качестве параметра. Этот класс “знает” все источники данных для нашего приложения. В начальном блоке viewmodel мы обновляем данные БД. Это делается вызовом метода обновления репозитория. А еще у viewmodel есть свойство data. Оно получает данные локально. Это гарантия, что у пользователя всегда будет что-то в интерфейсе, даже если устройство не в сети.

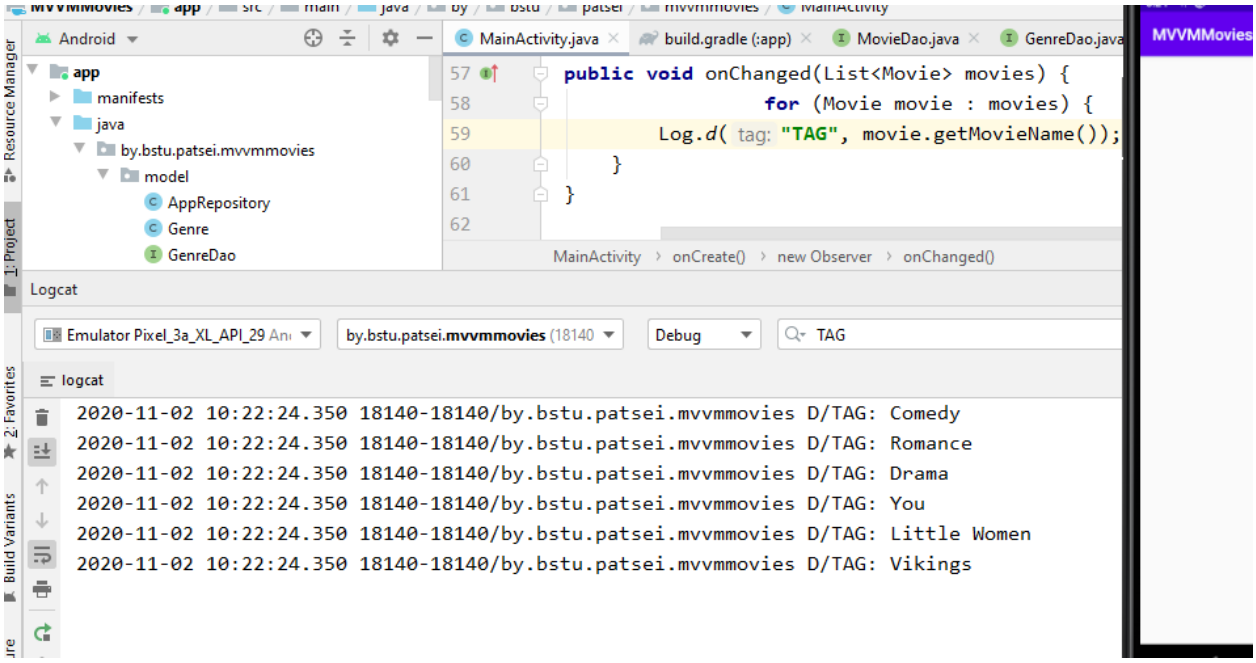
Переходим к активности и передадим даем ей ***ViewModel***

```
public class MainActivity extends AppCompatActivity {

    private MainActivityViewModel mainActivityViewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
      ...

mainActivityViewModel = new ViewModelProvider
        .AndroidViewModelFactory(getApplication())
        .create(MainActivityViewModel.class);
```

Теперь создадим все жарны и получим их. Они являются *LiveData:*

```
mainActivityViewModel = new ViewModelProvider
        .AndroidViewModelFactory(getApplication())
        .create(MainActivityViewModel.class);

mainActivityViewModel.getGenres().observe(this, new Observer<List<Genre>>() {
    @Override
    public void onChanged(List<Genre> genres) {
        for (Genre genre : genres) {
            Log.d("TAG", genre.getGenreName());
        }
    }
});

mainActivityViewModel.getGenreMovies(2).observe(this, new Observer<List<Movie>() {
    @Override
    public void onChanged(List<Movie> movies) {
                    for (Movie movie : movies) {
            Log.d("TAG", movie.getMovieName());
        }
    }
```

Запускаем, фильтруем log и видим что данные о жанрах и фильмах жанра 2 получаются.



Создадим **MainActivityClickHandlers** и метод для Fab.

```
public class MainActivity extends AppCompatActivity {
    ...
public class MainActivityClickHandlers{
    public void onFabClicked(View view){
        Toast.makeText(MainActivity.this, "Buuton", Toast.LENGTH_LONG);
    }
}
```

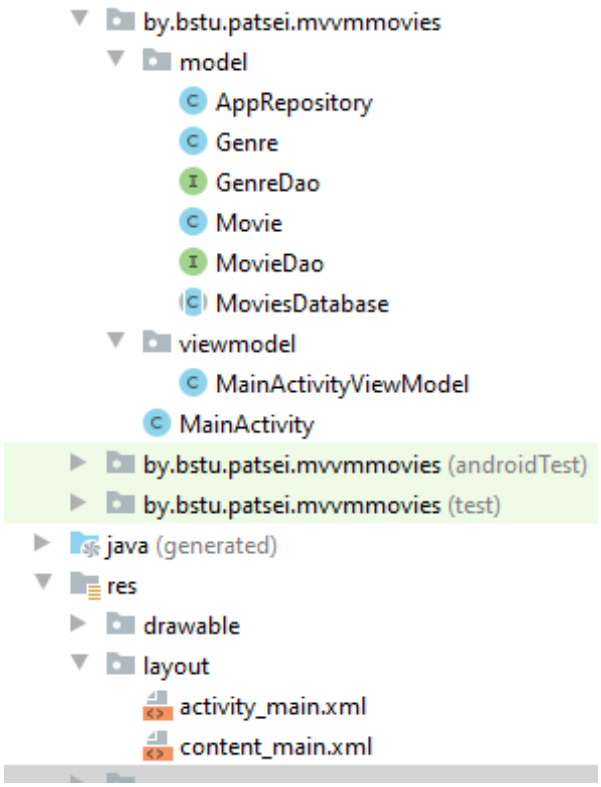
Перейдем в разметку *activity_main*.xml. Сделаем *layout* корневым элементом. Определим <*data*> и перменную:

```
<layout 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">
    <data>
        <variable
            name="clickHandlers"
            type="by.bstu.patsei.mvvmmovies.MainActivity.MainActivityClickHandlers"
/>
    </data>
    ...
<include layout="@layout/content_main" />

<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/fab"
    android:onClick="@{clickHandlers::onFabClicked}"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    android:layout_margin="@dimen/fab_margin"
    app:srcCompat="@android:drawable/ic_dialog_email" />
    ...
```

Создадим переменные классов Binding.

```
public class MainActivity extends AppCompatActivity {

    private MainActivityViewModel mainActivityViewModel;
    private ActivityMainBinding activityMainBinding;
    private  MainActivityClickHandlers clickHandlers;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        activityMainBinding = DataBindingUtil.setContentView(this,
                R.layout.activity_main);

        mainActivityViewModel = new ViewModelProvider
                .AndroidViewModelFactory(getApplication())
                .create(MainActivityViewModel.class);

        clickHandlers = new MainActivityClickHandlers();
        activityMainBinding.setClickHandlers(clickHandlers);
    ...
```

Уберем классы фрагментов и разметки фрагментов из проекта.

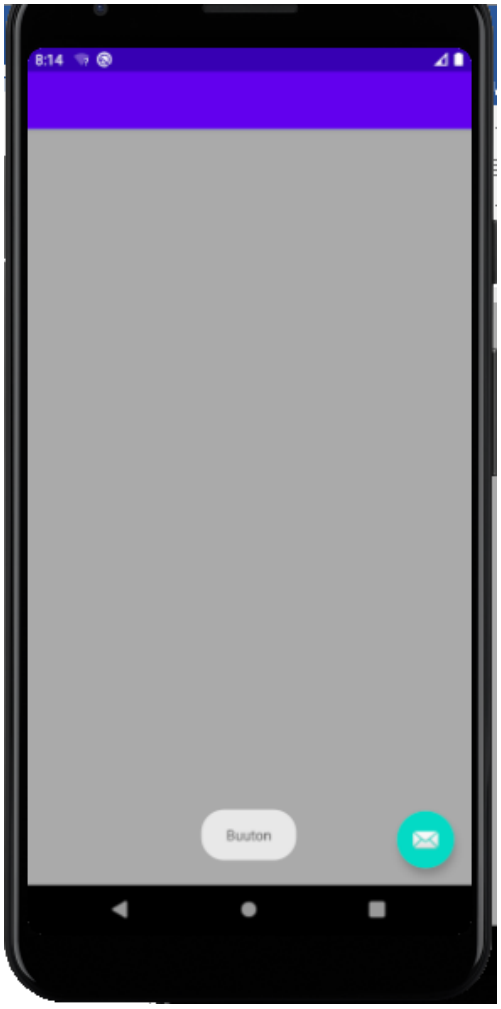


Поменяем content_main.xml следующим образом.

```
<?xml version="1.0" encoding="utf-8"?>
<layout 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">
```

```
<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior">

<!--    <fragment-->
<!--        android:id="@+id/nav_host_fragment"-->
<!--        android:name="androidx.navigation.fragment.NavHostFragment"-->
<!--        android:layout_width="0dp"-->
<!--        android:layout_height="0dp"-->
<!--        app:defaultNavHost="true"-->
<!--        app:layout_constraintBottom_toBottomOf="parent"-->
<!--        app:layout_constraintLeft_toLeftOf="parent"-->
<!--        app:layout_constraintRight_toRightOf="parent"-->
<!--        app:layout_constraintTop_toTopOf="parent"-->
<!--        app:navGraph="@navigation/nav_graph" />-->

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@android:color/darker_gray"
        app:layout_behavior="@string/appbar_scrolling_view_behavior"
        tools:context=".MainActivity"
        tools:showIn="@layout/activity_main"/>
</androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

Запустим и проверим как работает. Нажимем кнопку – видим Toast



Теперь добавим выпадающий списокк с жанрами. Внесм изменения в разметку *content_main.xml*. Добавим *Spinner*.

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/darker_gray"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".MainActivity"
    tools:showIn="@layout/activity_main">

   <Spinner
       android:id="@+id/Spinner"
       android:layout_width="400dp"
       android:layout_height="wrap_content"
       android:layout_margin="8dp"/>

</LinearLayout>
```

В *activity_main.xml* добавим переменную адаптер

```
<data>
    <variable
        name="clickHandlers"
        type="by.bstu.patsei.mvvmmovies.MainActivity.MainActivityClickHandlers" />

    <variable
        name="spinnerAdapter"
        type="android.widget.ArrayAdapter" />
</data>
```

Добавим namespace:

```
xmlns:bind ="http://schemas.android.com/apk/res-auto"
```

Будем передавать ссылки на переменные во вторичную разметку *content_main.xml*

```
<include layout="@layout/content_main"

    android:id="@+id/secondary_layout"
    bind:secondarySpinnerAdapter="@{spinnerAdapter}"
    bind:secondaryClickHandlers="@{clickHandlers}"
    />
```

Теперь достанем эти переменне во вторичной разметке *content_main.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<layout 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">
    <data>
        <variable
            name="secondarySpinnerAdapter"
            type="android.widget.ArrayAdapter" />
```

```
        <variable
            name="secondaryClickHandlers"
            type="by.bstu.patsei.mvvmmovies.MainActivity.MainActivityClickHandlers"
/>
    </data>
    <androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior">


    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@android:color/darker_gray"
        app:layout_behavior="@string/appbar_scrolling_view_behavior"
        tools:context=".MainActivity"
        tools:showIn="@layout/activity_main">

       <Spinner
           android:id="@+id/Spinner"
           android:layout_width="400dp"
           android:layout_height="wrap_content"
           android:layout_margin="8dp"/>


    </LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

Теперь получим жанры из выпадающего списка в *MainActivity*

```
public class MainActivity extends AppCompatActivity {

...
    private  Genre selectedGenre;


public class MainActivityClickHandlers {
...
public void onSelectedItem(AdapterView<?> parent, View view, int position, long id) {
    selectedGenre = (Genre) parent.getItemAtPosition(position);
    String message = "id - " + selectedGenre.getId() +
            "name - " + selectedGenre.getGenreName();
    Toast.makeText(MainActivity.this, message, Toast.LENGTH_LONG).show();
}
       ...
}
```

Связываем Spinner с адаптером

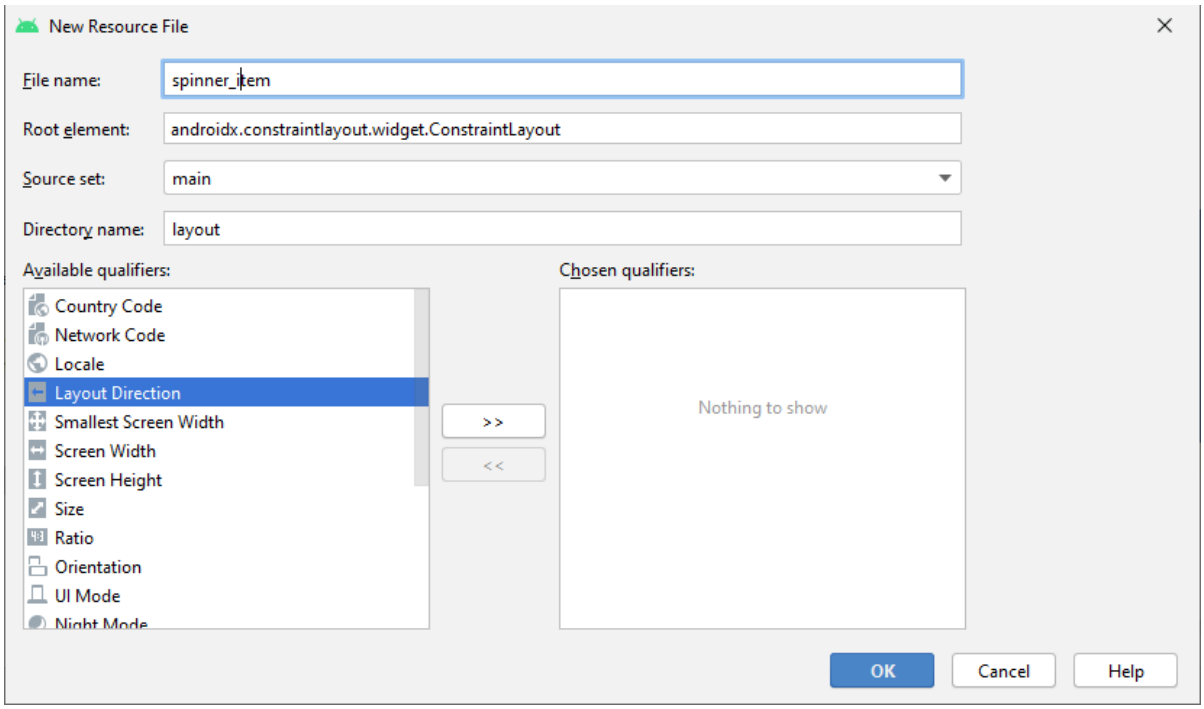
```
<Spinner
    android:id="@+id/Spinner"
    android:layout_width="400dp"
    android:layout_height="wrap_content"
    android:layout_margin="8dp"
    app:adapter = "@{secondarySpinnerAdapter}"
```

```
    android:onItemSelected = "@{secondaryClickHandlers::onSelectedItem}"
    />
```

Опеределяем в активности список жанров

```
public class MainActivity extends AppCompatActivity {

...
    private ArrayList<Genre> genreArrayList;
    ...
mainActivityViewModel.getGenres().observe(this, new Observer<List<Genre>>() {
    @Override
    public void onChanged(List<Genre> genres) {
      genreArrayList = (ArrayList<Genre>) genres;

        for (Genre genre : genres) {
            Log.d("TAG", genre.getGenreName());
        }
        showInSpinner();

    }
})
```

Добавим layout для отображения spinner – spinner_item.xml



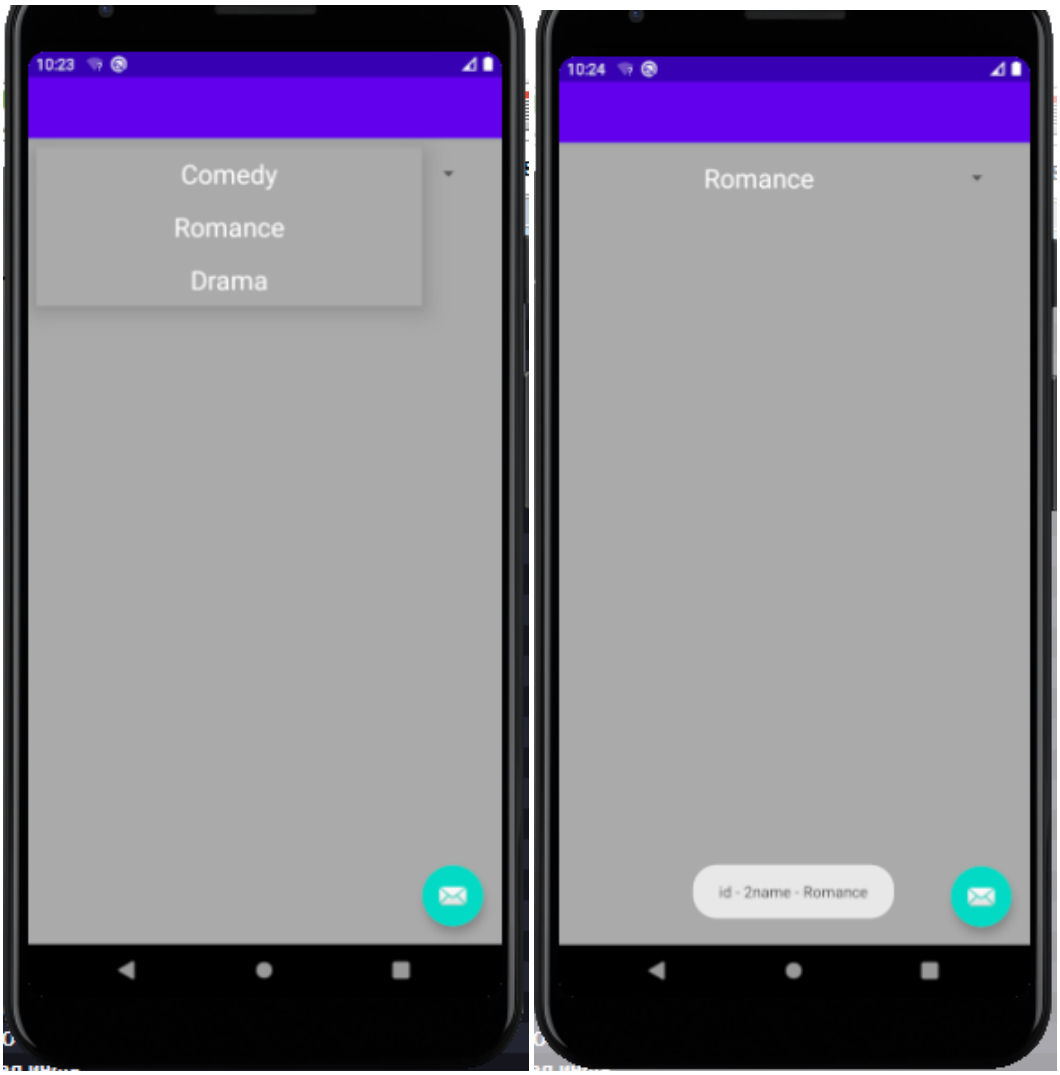
Поменяем разметку – там будет TextView для одного типа жанра.

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="24sp"
    android:background="@android:color/darker_gray"
    android:gravity="center"
    android:layout_margin="2dp"
    android:textColor="@android:color/white"
    android:padding="8dp"/>
```

Определяем метод *showInSpinner*(); в классе активности

```
private void showInSpinner() {

    ArrayAdapter<Genre> genreArrayAdapter = new ArrayAdapter<Genre>(this,
            R.layout.spinner_item, genreArrayList);
    genreArrayAdapter.setDropDownViewResource(R.layout.spinner_item);
    activityMainBinding.setSpinnerAdapter(genreArrayAdapter);

}
```

Запустим и проверим:

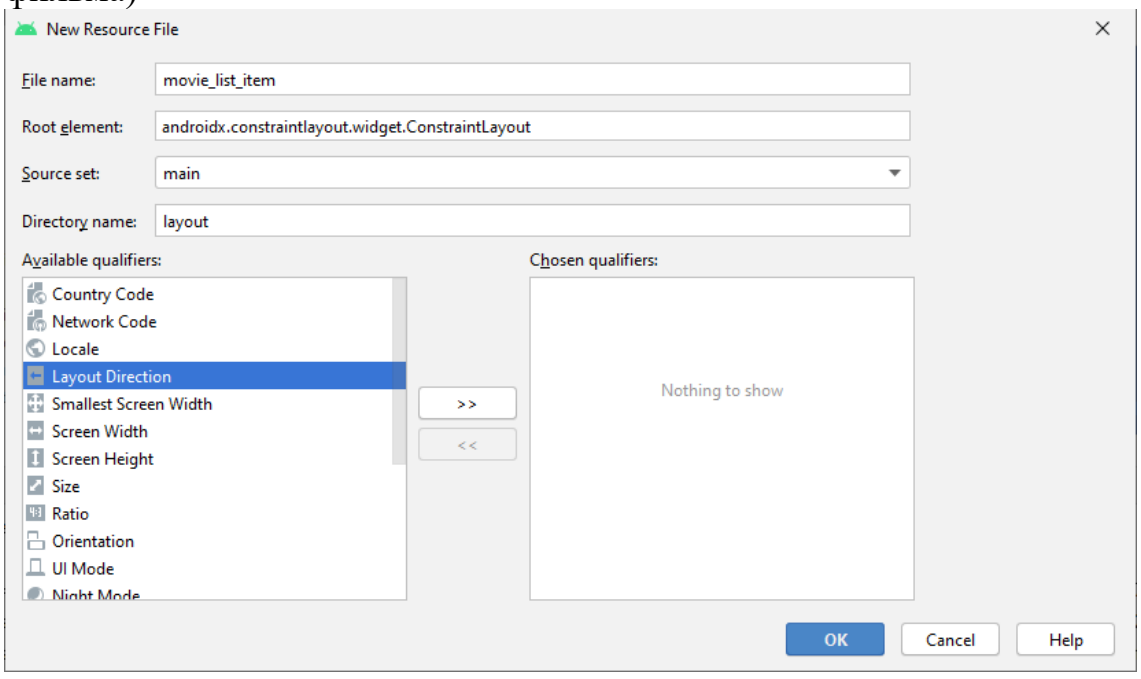


Слеюующий шаг - создадим *RecyclerView* для вывода списка фильмов Добавим егео после Spinner в *content_main.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<layout 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">

    <data>
        <variable
            name="secondarySpinnerAdapter"
            type="android.widget.ArrayAdapter" />
```

```
        <variable
            name="secondaryClickHandlers"
            type="by.bstu.patsei.mvvmmovies.MainActivity.MainActivityClickHandlers"
/>
    </data>
    <androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@android:color/darker_gray"
        app:layout_behavior="@string/appbar_scrolling_view_behavior"
        tools:context=".MainActivity"
        tools:showIn="@layout/activity_main">

       <Spinner
           android:id="@+id/Spinner"
           android:layout_width="400dp"
           android:layout_height="wrap_content"
           android:layout_margin="8dp"
           app:adapter = "@{secondarySpinnerAdapter}"
           android:onItemSelected = "@{secondaryClickHandlers::onSelectedItem}"
           />
        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/recyclerView"
            android:layout_margin="8dp"
            android:layout_width="match_parent"
            android:layout_height="match_parent"/>

    </LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

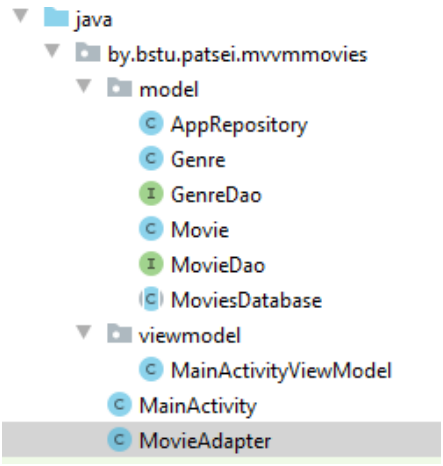
Добавим новый layout - *movie_list_item.xml* (отображение одного фильма)



Там будет layout с переменной *movie*, с *CardView*, с *TextView* для вывода фильма и data binding.

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

    <data>

        <variable
            name="movie"
            type="by.bstu.patsei.mvvmmovies.model.Movie" />

    </data>

    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="2dp"
        app:cardCornerRadius="4dp">

        <LinearLayout
            android:orientation="vertical"
            android:padding="8dp"
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <TextView
                android:id="@+id/nameTextView"
                android:text="@{movie.movieName}"
                android:textSize="24sp"
                android:textStyle="bold"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"/>

            <TextView
                android:id="@+id/descriptionTextView"
                android:text="@{movie.movieDescription}"
                android:textSize="24sp"
                android:textStyle="bold"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"/>

        </LinearLayout>
```

Затем надо создать адаптер для *RecycleView* - ***MovieAdapter***.




```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import java.util.ArrayList;
import androidx.annotation.NonNull;
import androidx.databinding.DataBindingUtil;
import androidx.recyclerview.widget.RecyclerView;
import by.bstu.patsei.mvvmmovies.databinding.MovieListItemBinding;
import by.bstu.patsei.mvvmmovies.model.Movie;

public class MovieAdapter extends RecyclerView.Adapter<MovieAdapter.MovieViewHolder>
{

    private OnItemClickListener onItemClickListener;
    private ArrayList<Movie> movieArrayList = new ArrayList<>();

    @NonNull
    @Override
    public MovieViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
        MovieListItemBinding movieListItemBinding = DataBindingUtil.inflate(
                LayoutInflater.from(parent.getContext()), R.layout.movie_list_item,
                parent, false
        );
        return new MovieViewHolder(movieListItemBinding);
    }

    @Override
    public void onBindViewHolder(@NonNull MovieViewHolder holder, int position) {
        Movie movie = movieArrayList.get(position);
        holder.movieListItemBinding.setMovie(movie);
    }

    @Override
    public int getItemCount() {
        return movieArrayList.size();
    }

    class MovieViewHolder extends RecyclerView.ViewHolder{

        MovieListItemBinding movieListItemBinding;

        public MovieViewHolder(@NonNull MovieListItemBinding movieListItemBinding) {
            super(movieListItemBinding.getRoot());
            this.movieListItemBinding = movieListItemBinding;
            movieListItemBinding.getRoot().setOnClickListener(new
View.OnClickListener() {
```

```
                @Override
                public void onClick(View view) {
                    int position = getAdapterPosition();
                    if (onItemClickListener != null && position !=
RecyclerView.NO_POSITION) {

onItemClickListener.onItemClick(movieArrayList.get(position));
                    }
                }
            });
        }
    }

    public interface OnItemClickListener {
        void onItemClick(Movie movie);
    }

    public void setOnItemClickListener(OnItemClickListener onItemClickListener) {
        this.onItemClickListener = onItemClickListener;
    }

    public void setMovieArrayList(ArrayList<Movie> movieArrayList) {
        this.movieArrayList = movieArrayList;
        notifyDataSetChanged();
    }
}
```

В активности:

```
private ArrayList<Movie> movieArrayList;
private RecyclerView recyclerView;
private MovieAdapter movieAdapter;
private int selectedMovieId;
```

И напишем метод вывода фильмов принадлежащих жанру:

```
    private     void     loadGenreMoviesInArrayList(int     genreId)     {

    mainActivityViewModel.getGenreMovies(genreId).observe(this,           new
Observer<List<Movie>>() {
      @Override
        public void onChanged(List<Movie> movies) {
            movieArrayList = (ArrayList<Movie>) movies;
            loadRecyclerView();
        }
    });
}
```

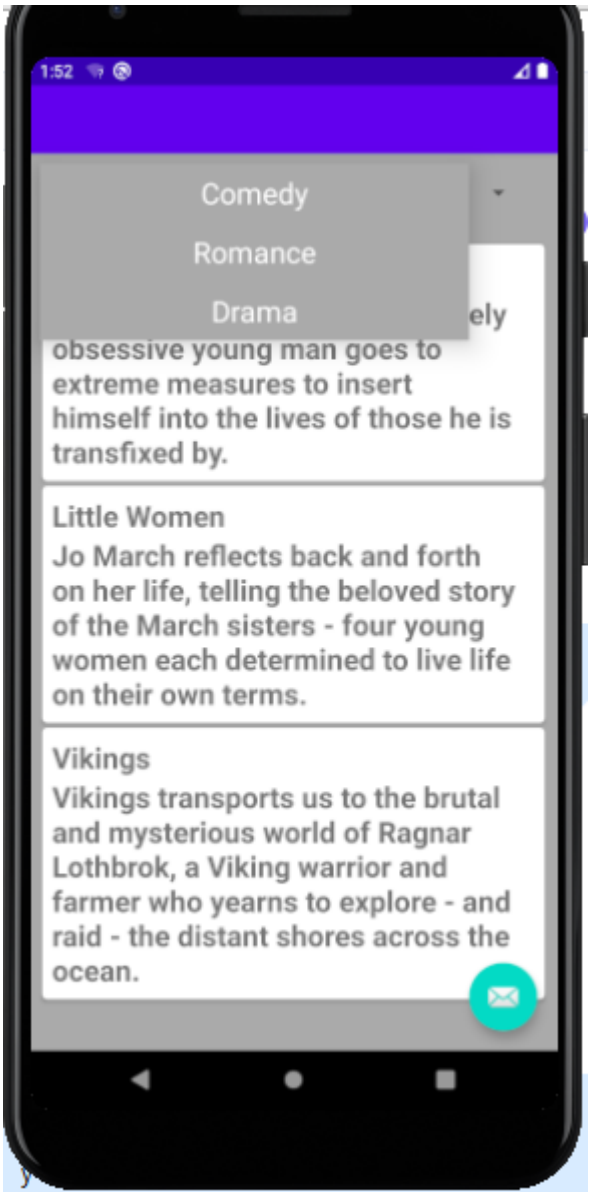
И вызовем его

```
    public void onSelectedItem(AdapterView<?> parent, View view, int position, long
id) {
        selectedGenre = (Genre) parent.getItemAtPosition(position);
        String message = "id - " + selectedGenre.getId() +
                "name - " + selectedGenre.getGenreName();
        Toast.makeText(MainActivity.this, message, Toast.LENGTH_LONG).show();
        loadGenreMoviesInArrayList(selectedGenre.getId());
    }
}
```

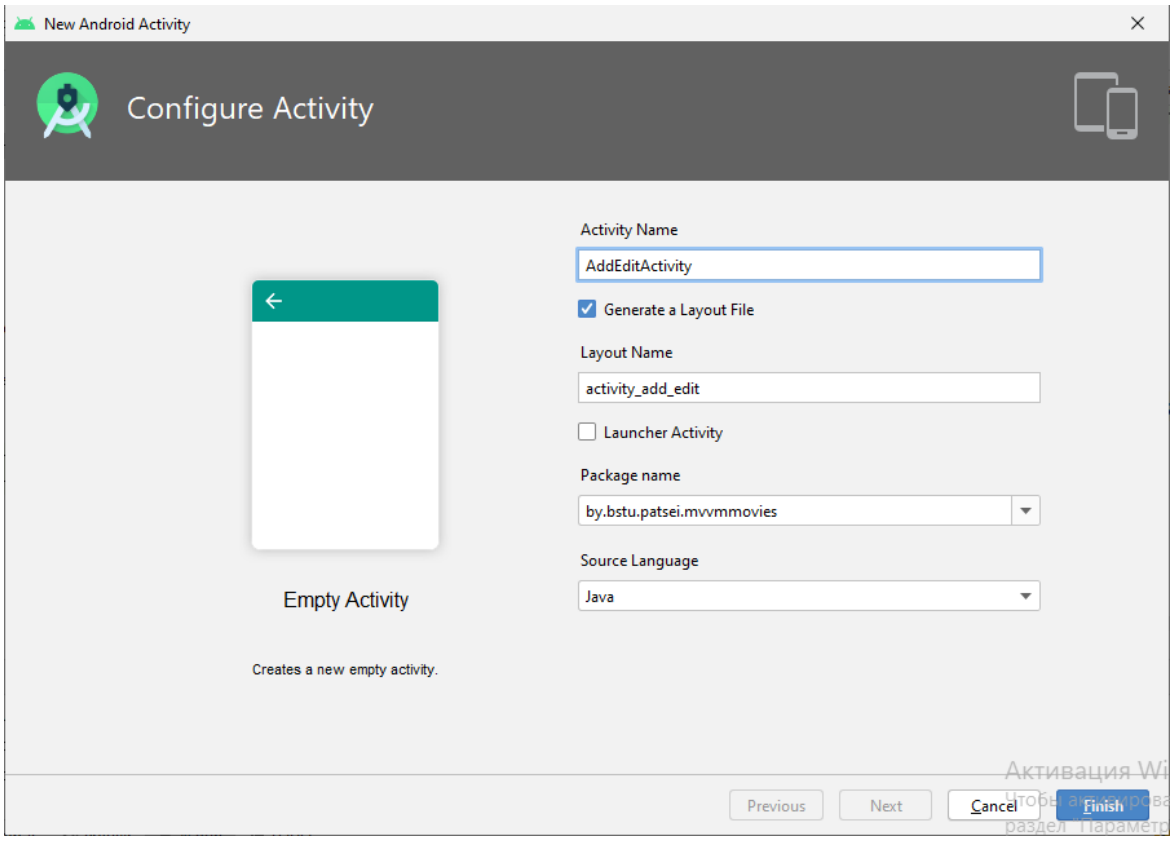
Пишем метод для загрузки списка в recycleview

```
private void loadRecyclerView() {

    recyclerView = activityMainBinding.secondaryLayout.recyclerView;
    recyclerView.setLayoutManager(new LinearLayoutManager(this));
    recyclerView.setHasFixedSize(true);

    movieAdapter = new MovieAdapter();
    movieAdapter.setMovieArrayList(movieArrayList);
    recyclerView.setAdapter(movieAdapter);
      }
```

Проверяем. Все отображается по жанрам.



Осталось добавить CRUD операции. Для этого создадаим новую ***AddEditActivity***.



Поменяем разметку activity_add_edit.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout 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">

    <data>

        <variable
            name="clickHandlers"

type="by.bstu.patsei.mvvmmovies.AddEditActivity.AddEditActivityClickHandlers" />

        <variable
            name="movie"
            type="by.bstu.patsei.mvvmmovies.model.Movie" />

    </data>

    <LinearLayout
        android:orientation="vertical"
        android:padding="4dp"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".AddEditActivity">

        <TextView
            android:id="@+id/movieNameTextView"
            android:text="Name"
            android:layout_marginTop="16dp"
            android:textSize="24sp"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>
```

```
        <EditText
            android:id="@+id/movieNameEditText"
            android:hint="Name"
            android:text="@={movie.movieName}"
            android:layout_marginTop="16dp"
            android:textSize="24sp"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>

        <TextView
            android:id="@+id/movieDescriptionTextView"
            android:text="Description"
            android:layout_marginTop="16dp"
            android:textSize="24sp"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>

        <EditText
            android:id="@+id/movieDescriptionEditText"
            android:hint="Description"
            android:text="@={movie.movieDescription}"
            android:layout_marginTop="16dp"
            android:textSize="24sp"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>


        <Button
            android:id="@+id/okButton"
            android:onClick="@{clickHandlers::onOkButtonClicked}"
            android:text="Ok"
            android:textSize="24sp"
            android:layout_marginTop="48dp"
            android:gravity="center"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>

    </LinearLayout>

</layout>
```

Весь код проекта в репозитории
https://github.com/PatseiBSTU/MVVMMovies.git

И еще немного

Существует библиотека AndroidMvvmHelper https://github.com/stfalcon-studio/AndroidMvvmHelper - это набор базовых классов для удобной работы с MVVM. В этот перечень входят классы как для работы с Activity (BindingActivity и ActivityViewModel), так и с Fragment (BindingFragment и FragmentViewModel), в которых уже реализована логика связывания, а также определены необходимые методы для получения callback-ов. Для того, чтобы начать ее использовать - необходимо просто определить зависимость в grdale-файле:

```
dependencies {
   ...
   compile 'com.github.stfalcon:androidmvvmhelper:X.X'
}
```

Хотя решение с библиотекой и упрощает жизнь, создание классов все еще достаточно трудоемкий процесс. Для решения этой задачи есть плагин для IntelliJ IDEA и Android Studio - **MVVM Generator**. Он позволяет создать класс *BindingActivity* (или *BindingFragment*), его *ViewModel*, подготовленный xml-файл для разметки и зарегистрировать компонент в *AndroidManifest* (в случае активности). Кроме того, если плагин не обнаружит зависимости библиотеки ***MVVMHelper***, она будет автоматически добавлена.

Чем больше кода имеет под собой хорошую архитектуру, тем проще поддерживать и тестировать приложение. Вот почему важно пользоваться паттернами. С ними проще создать стабильно работающие программы

https://www.fandroid.info/lektsiya-8-po-arhitekture-android-data-binding-mvvm/