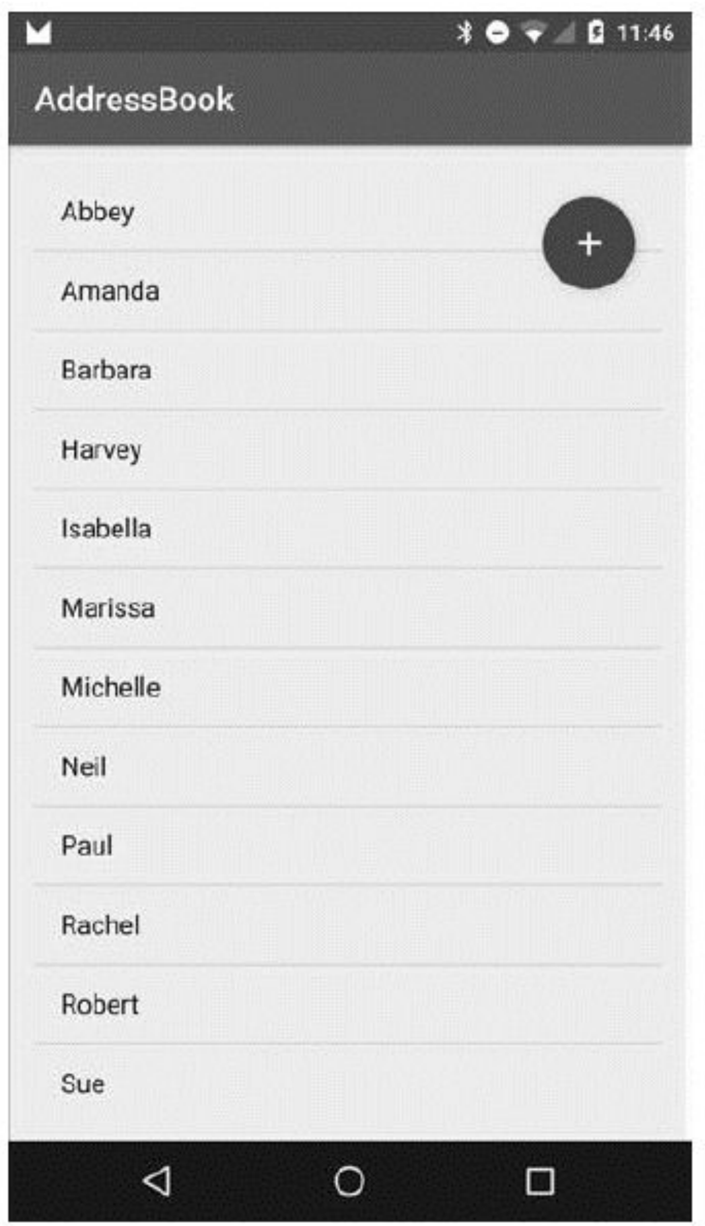
Android 6
Работа с базой данных SQLite

Рассматриваемые вопросы

- Использование транзакций и стека возврата для динамического присоединения и отсоединения фрагментов
- Использование RecyclerView для вывода информации из базы данных
- Создание и открытие баз данных SQLite с помощью класса SQLiteOpenHelper
- Использование классов ContentProvider и SQLiteDatabase для работы с информацией в базе данных SQLite
- Использование класса ContentResolver для вызова методов ContentProvider при выполнении операций с базой данных
- Использование классов LoaderManager и Loader для асинхронных обращений к базе данных за пределами потока графического интерфейса
- Работа с результатами запроса базы данных с использованием класса Cursor
- Определение стилей с часто используемыми значениями и атрибутами GUI, применяемыми к разным компонентам графического интерфейса

Целевое приложение



Целевое приложение

4:06

AddressBook

Abbey
Amanda
Barbara
Harvey
Isabella
Marissa
Michelle
Neil
Paul
Rachel
Robert
Sue

+

Name: Paul
Phone: 555-555-5555
E-Mail: paul@bug2bug.com
Street: 1 Main Street
City: Boston
State: MA
Zip: 02215

Используемые возможности

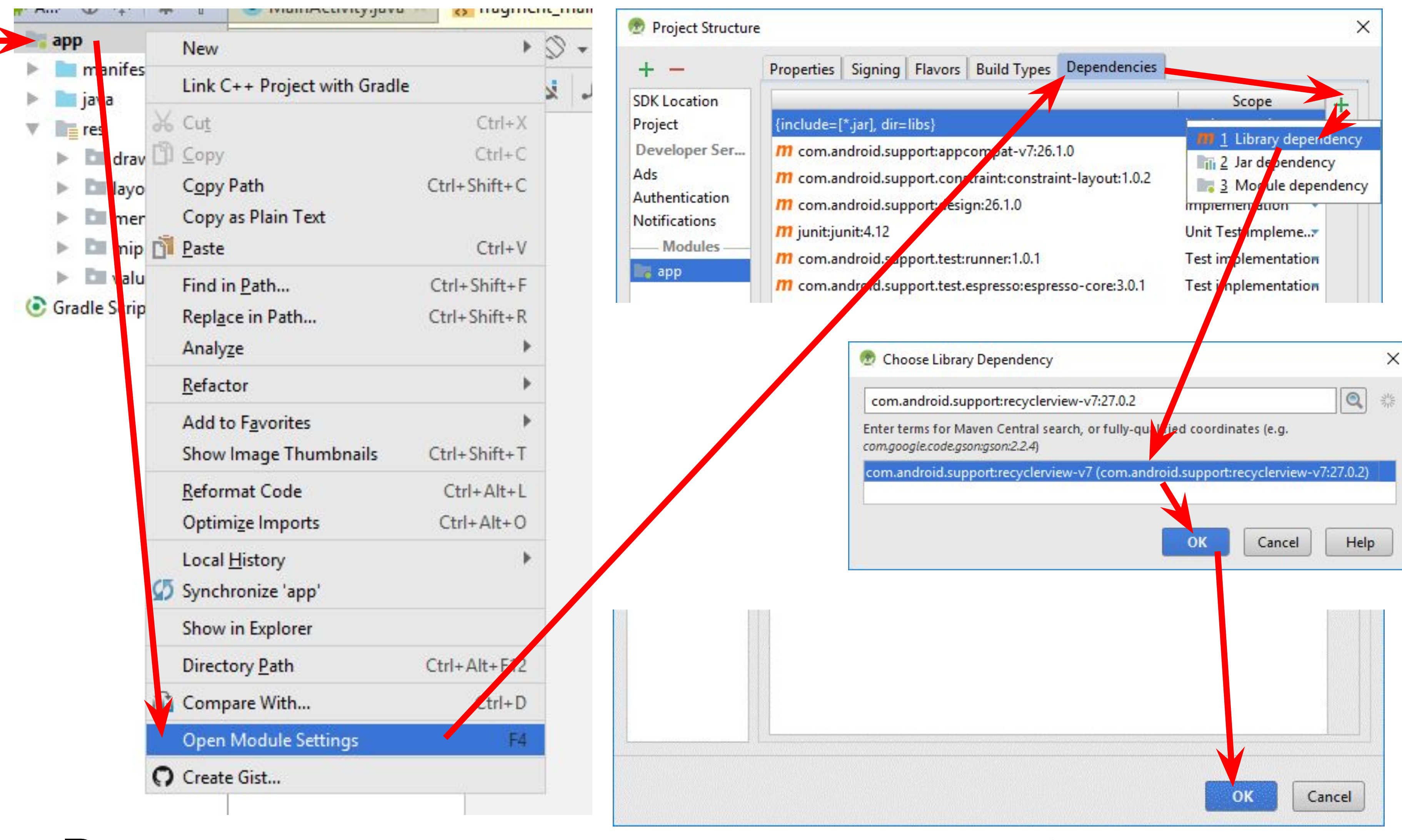
- динамическое отображение фрагментов с помощью **FragmentManager** и **FragmentTransaction**, использование стека возврата фрагментов (кнопка)
- передача данных между фрагментами и активностями (методы обратного вызова и интерфейс **Serializable**)
- компонент **RecyclerView** для вывода списка данных
- работа с базой данных SQLite (классы **SQLiteOpenHelper**, **SQLiteDatabase**)
- классы **ContentProvider** и **ContentResolver** (для открытия доступа к данным приложения и выполнения асинхронных операций с базой данных за пределами потока GUI)
- асинхронные операции с базами данных (классы **Loader** и **LoaderManager**)
- стили компонентов графического интерфейса пользователя
- определение фона для компонентов TextView

Создание проекта

- Имя проекта: L5 AddrBook
- Minimum SDK: API 23: Android 6.0 (Marshmallow)
- Шаблон: **Basic Activity**
- Флажок Use a Fragment
- Добавить значок в проект
- Настроить поддержку Java SE 7 (см. лекцию 3)

Создание проекта

- Добавить библиотеку RecyclerView



- В colors.xml задать colorAccent=#FF4081 (если это не так)

Создание классов приложения

- Переименовать фрагмент главной

ook] - [app] - ...\app\src\main\java\com\example\someone\l5addrbook\MainActivityFragment.jav

Build Run Tools VCS Window Help

app

com example someone l5addrbook MainActivityFragment

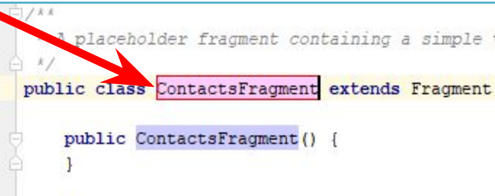
inActivity.java fragment_main.xml MainActivityFragment.java

MainActivityFragment

```
package com.example.someone.l5addrbook;

import ...

/**
 * A placeholder fragment containing a simple view.
 */
public class MainActivityFragment

    public MainActivityFragment()
    }

    @Override
    public View onCreateView(Layo
                             Bund
        return inflater.inflate(
    }
}
```

Copy Reference	Ctrl+Alt+Shift+C
Paste	Ctrl+V
Paste from History...	Ctrl+Shift+V
Paste Simple	Ctrl+Alt+Shift+V
Column Selection Mode	Alt+Shift+Insert
Find Usages	Alt+F7
Find Sample Code	Alt+F8
Refactor	

Rename...	Shift+F6
Change Signature...	Ctrl+F6
Type Migration...	Ctrl+Shift+F6
Make Static...	
Convert To Instance Method...	
Move...	F6
Copy...	F5
Safe Delete...	Alt+Delete
Extract	
Inline...	Ctrl+Alt+N
Find and Replace Code Duplicates...	
Invert Boolean...	
Pull Members Up...	
Push Members Down...	
Use Interface Where Possible...	
Replace Inheritance with Delegation...	
Remove Middleman...	
Wrap Method Return Value...	
Convert Anonymous to Inner...	
Encapsulate Fields...	
Replace Temp with Query...	



+Enter

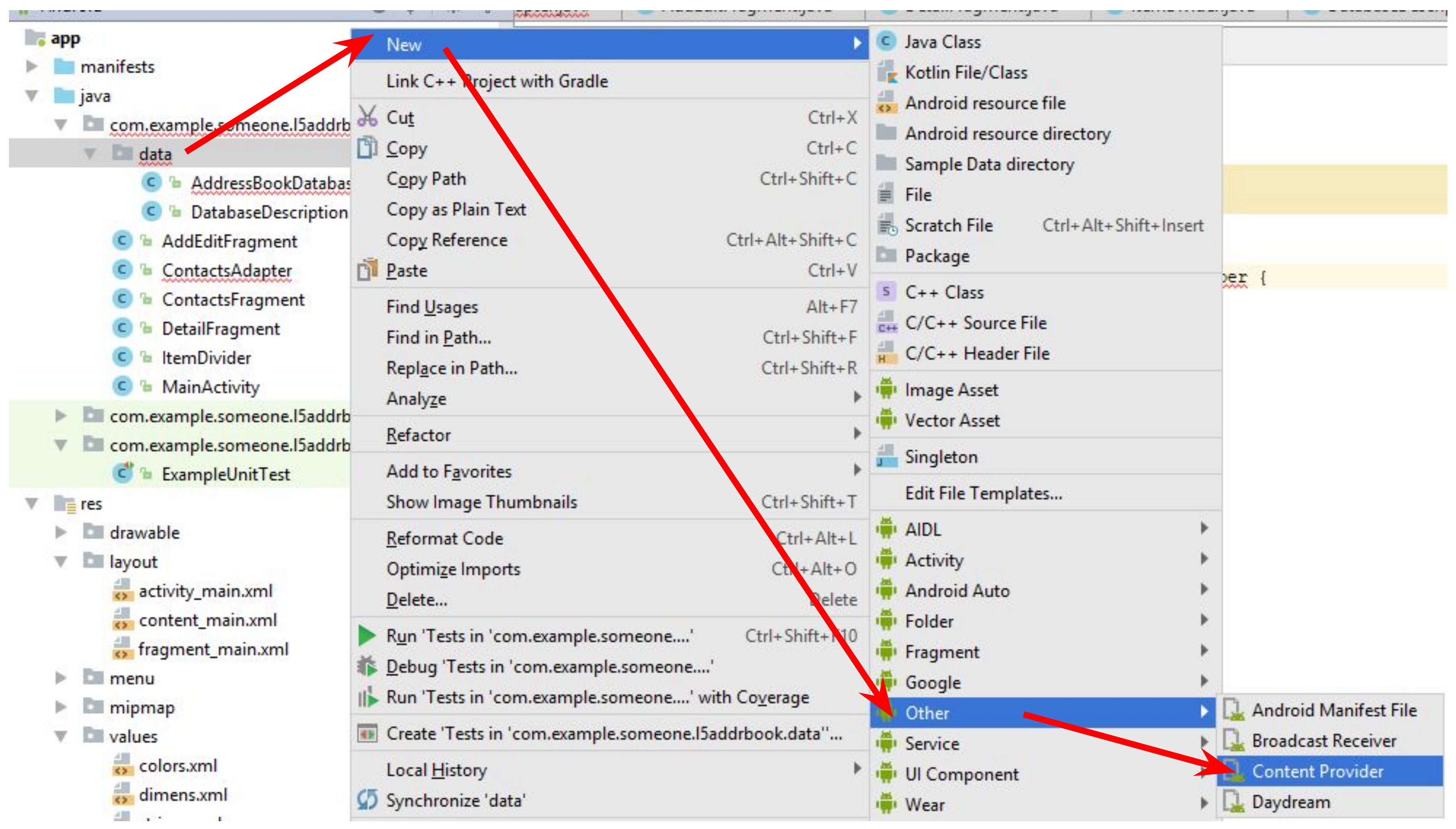
Создание классов приложения

- В основном пакете создать классы:
  - **ContactsAdapter** — субкласс RecyclerView.Adapter, поставляющий данные компоненту RecyclerView класса ContactsFragment;
  - **AddEditFragment** — субкласс android.support.v4.app.Fragment, предоставляющий графический интерфейс для добавления нового или редактирования существующего фрагмента;
  - **DetailFragment** — субкласс android.support.v4.app.Fragment, отображающий информацию одного контакта и предоставляющий команды меню для редактирования и удаления этого контакта;
  - **ItemDivider** — субкласс RecyclerView.ItemDecoration, используемый компонентом RecyclerView класса ContactsFragment для рисования горизонтальной линии между элементами.

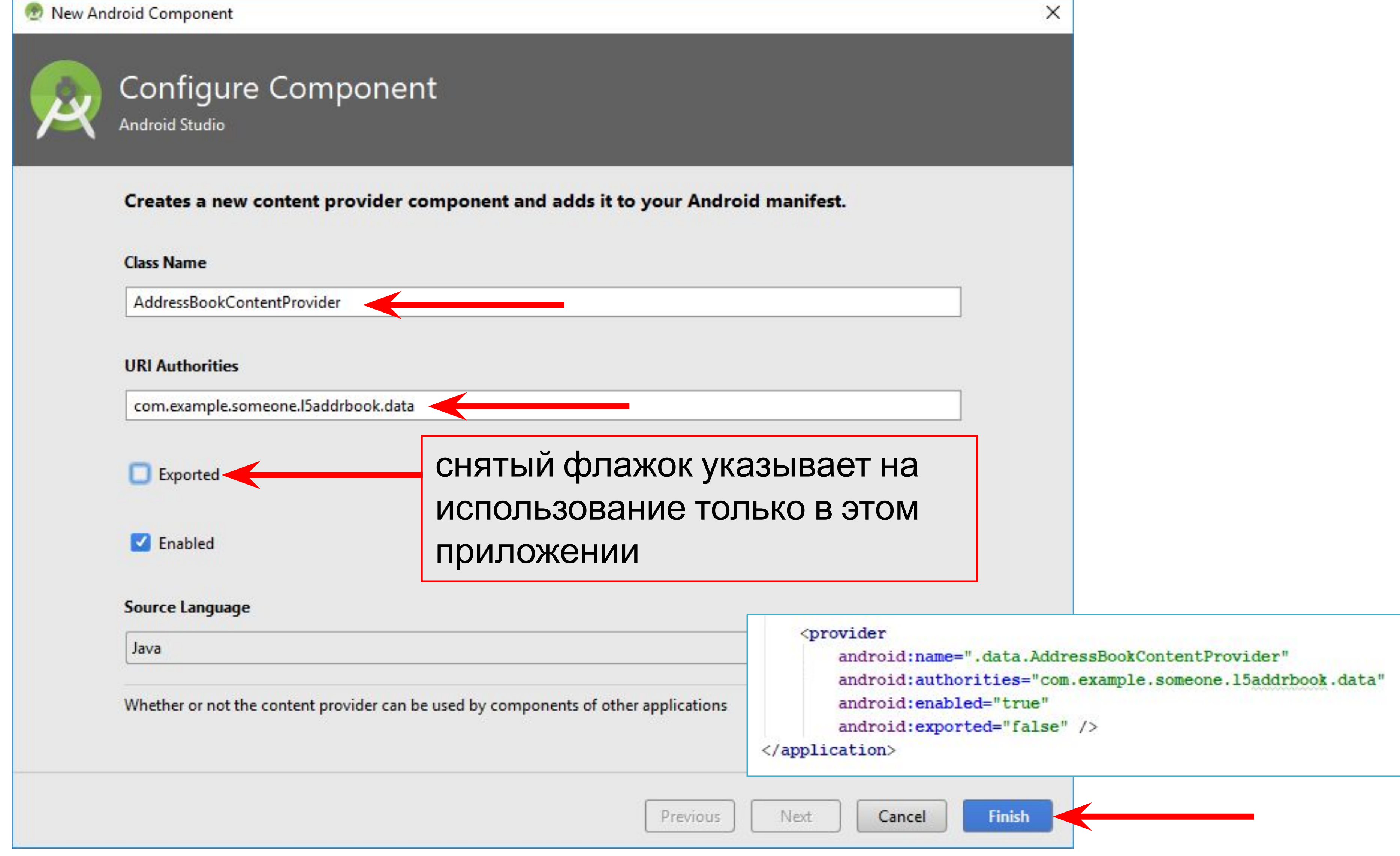
Создание классов приложения

- Там же создать вложенный пакет **data** (New→Package)
- Создать классы в пакете data:
  - **DatabaseDescription** содержит описание таблицы **contacts** базы данных.
  - **AddressBookDatabaseHelper** (субкласс SQLiteOpenHelper) создает базу данных и используется для работы с ней.
  - **AddressBookContentProvider** (субкласс ContentProvider) определяет, как приложение будет работать с базой данных.

Создание классов приложения (AddressBookContentProvider)

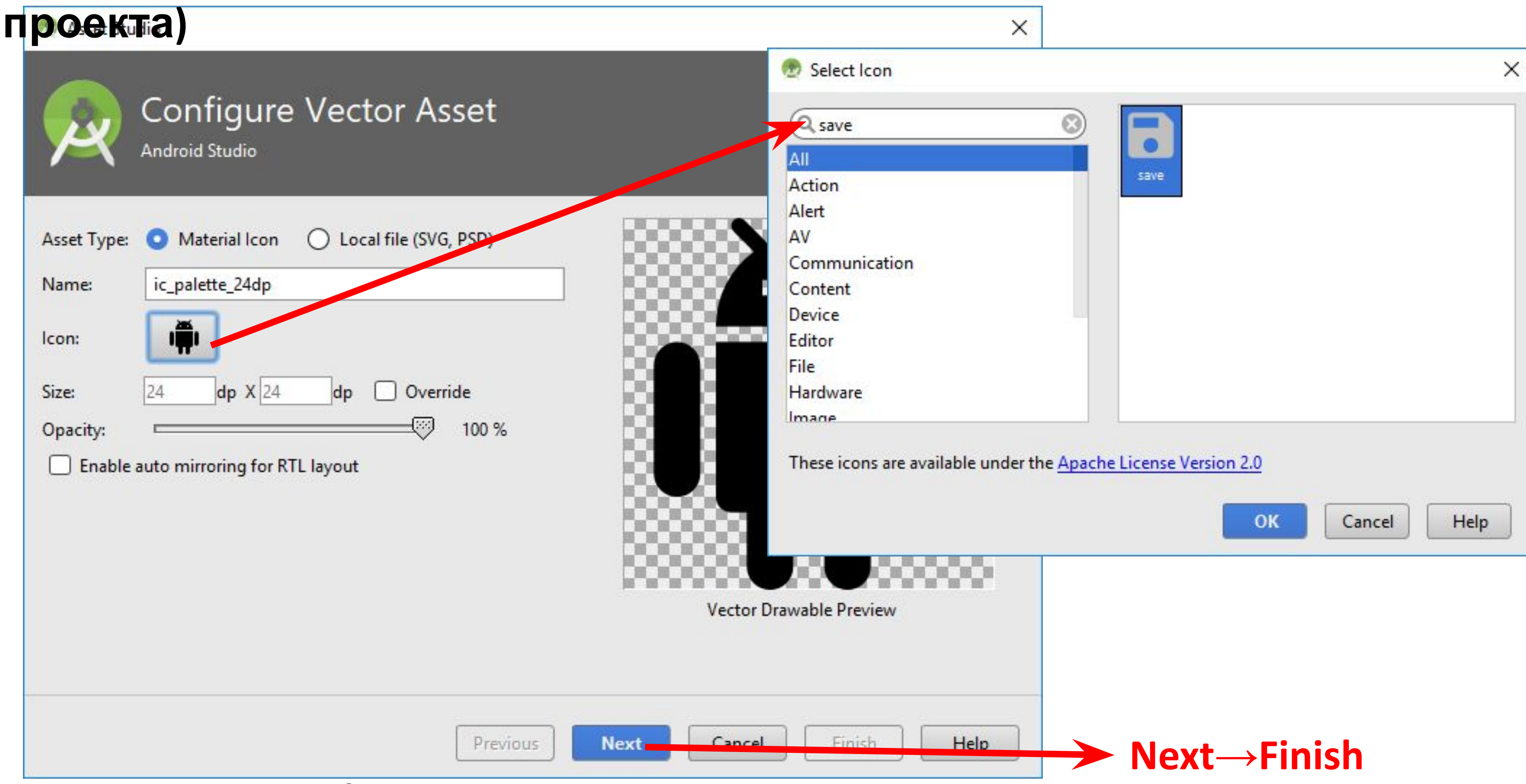


Создание классов приложения (AddressBookContentProvider)



Импорт необходимых значков

File→New→Vector Asset (при активной корневой папке проекта)



Аналогично добавить ресурсы: add, edit, delete

В XML-файлах заменить значение fillColor на @android:color/white

Переименовать XML-файлы, убрав из имени «black_» (если это есть в имени)

Определение строковых ресурсов

Имя ресурса	Значение
menuitem_edit	Edit
menuitem_delete	Delete
hint_name_required	Name (Required)
hint_email	E-Mail
hint_phone	Phone
hint_street	Street
hint_city	City
hint_state	State
hint_zip	Zip
label_name	Name:
label_email	E-Mail:
label_phone	Phone:
label_street	Street:

Определение строковых ресурсов

Имя ресурса	Значение
label_city	City:
label_state	State:
label_phone	Phone:
label_street	Street:
label_city	City:
label_state	State:
label_zip	Zip:
confirm_title	Are You Sure?
confirm_message	This will permanently delete the contact
button_cancel	Cancel
button_delete	Delete
contact_added	Contact added successfully

Определение строковых ресурсов

Имя ресурса	Значение
contact_not_added	Contact was not added due to an error
contact_updated	Contact updated
contact_not_updated	Contact was not updated due to an error
invalid_query_uri	Invalid query Uri:
invalid_insert_uri	Invalid insert Uri:
invalid_update_uri	Invalid update Uri:
invalid_delete_uri	Invalid delete Uri:
insert_failed	Insert failed: s

Стили для описания контакта

ent_main.xml × strings.xml × styles.xml ×

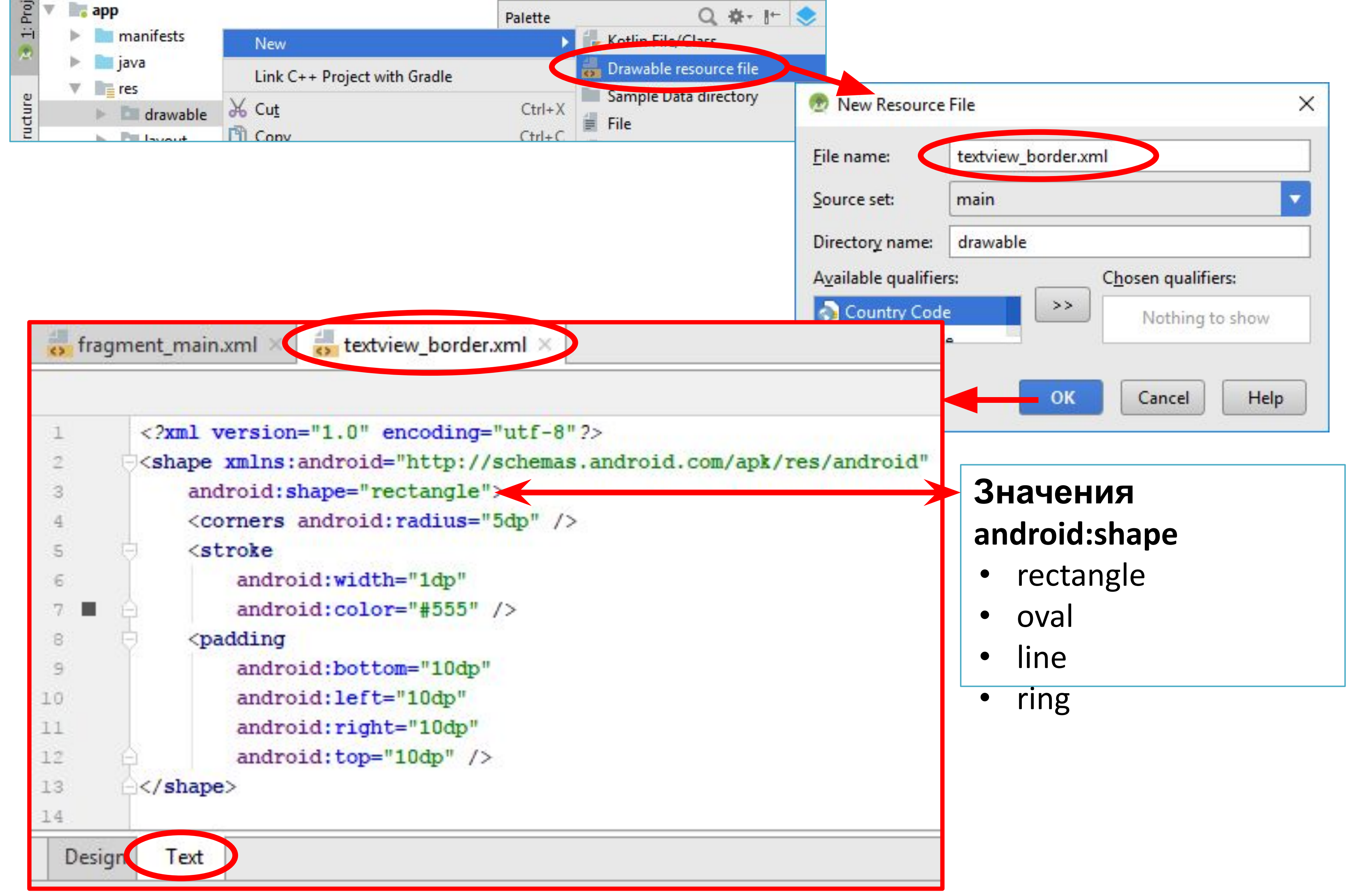
emes in the project in the theme editor.

```
<style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />

<style name="ContactLabelTextView">
    <item name="android:layout_width">wrap_content</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:layout_gravity">right|center_vertical</item>
</style>

<style name="ContactTextView">
    <item name="android:layout_width">wrap_content</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:layout_gravity">fill_horizontal</item>
    <item name="android:textSize">16sp</item>
    <item name="android:background">@drawable/textview_border</item>
</style>
</resources>
```

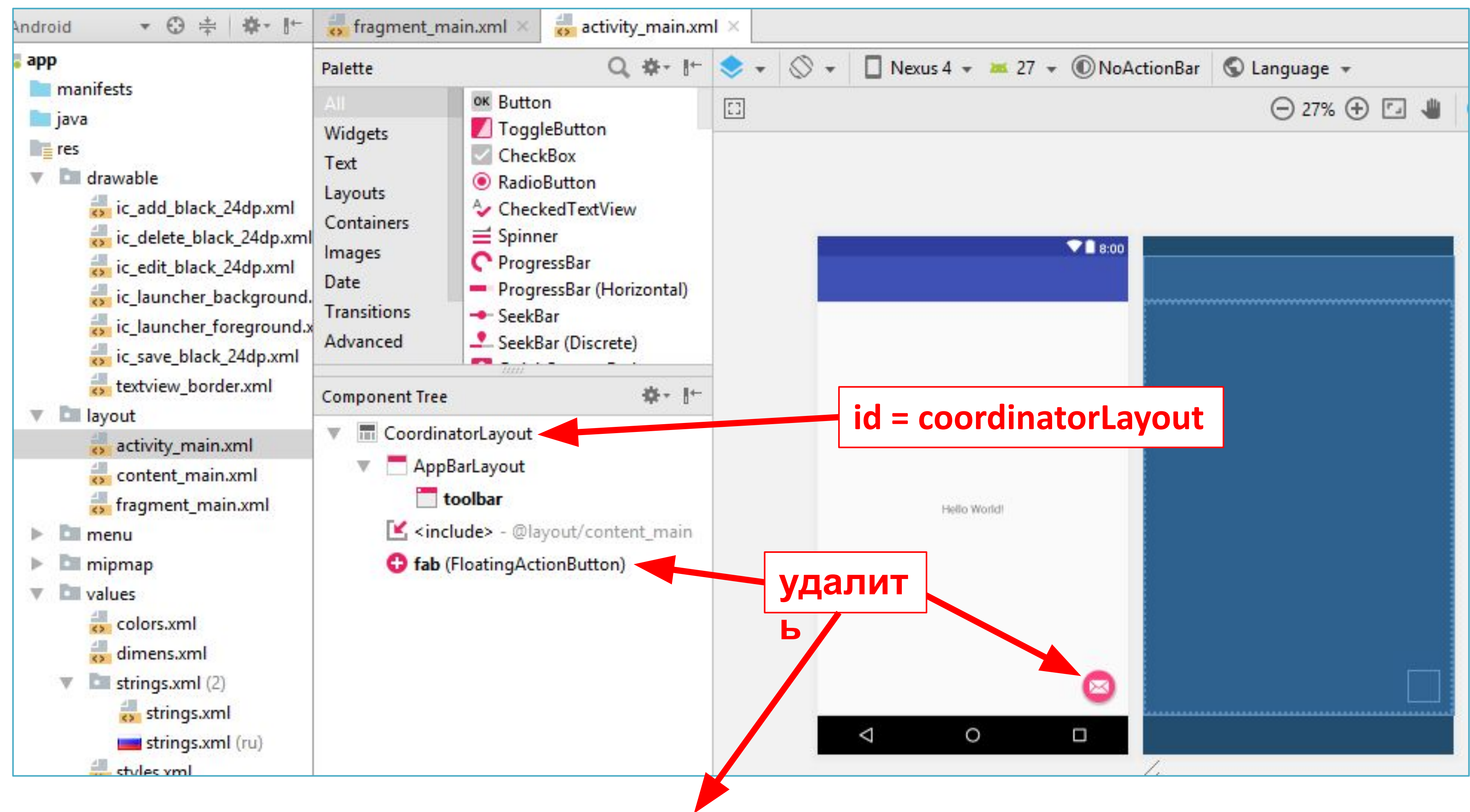
стиль для подписей полей контакта

стиль для значений полей контакта

Описание ресурса фигуры

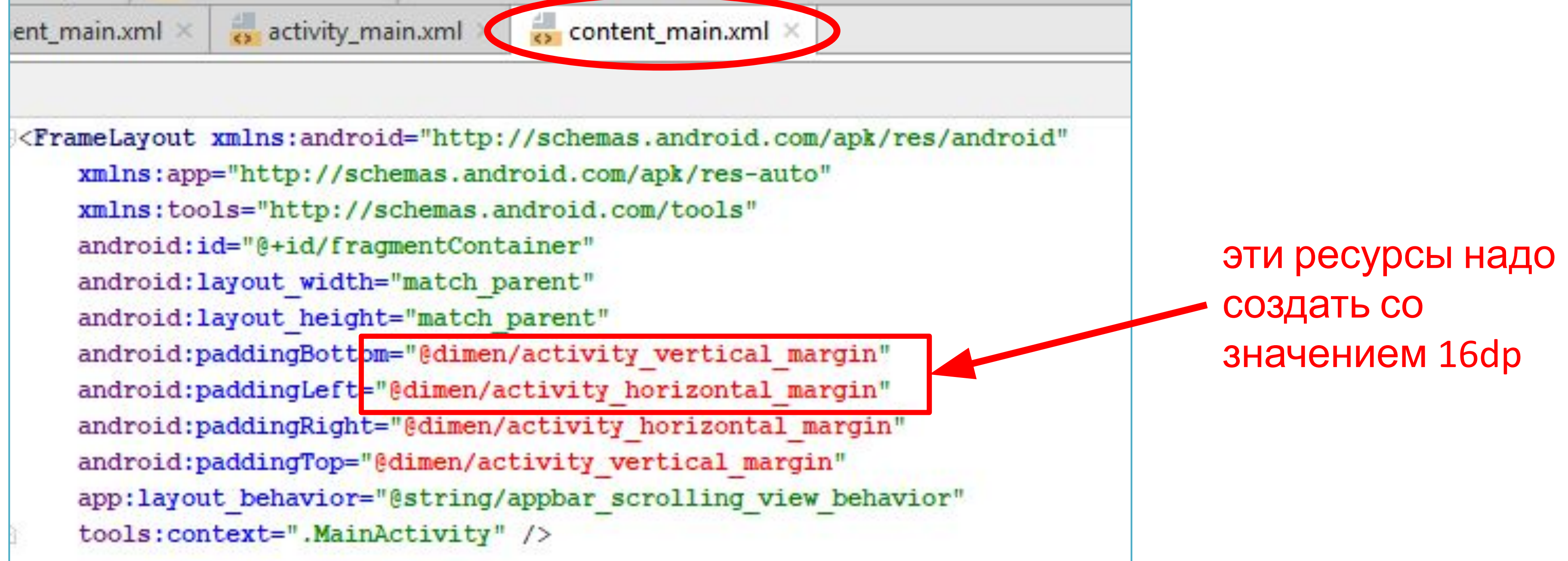


Построение графического интерфейса. Макет MainActivity



код настройки `FloatingActionButton` из метода `onCreate()` класса `MainActivity`

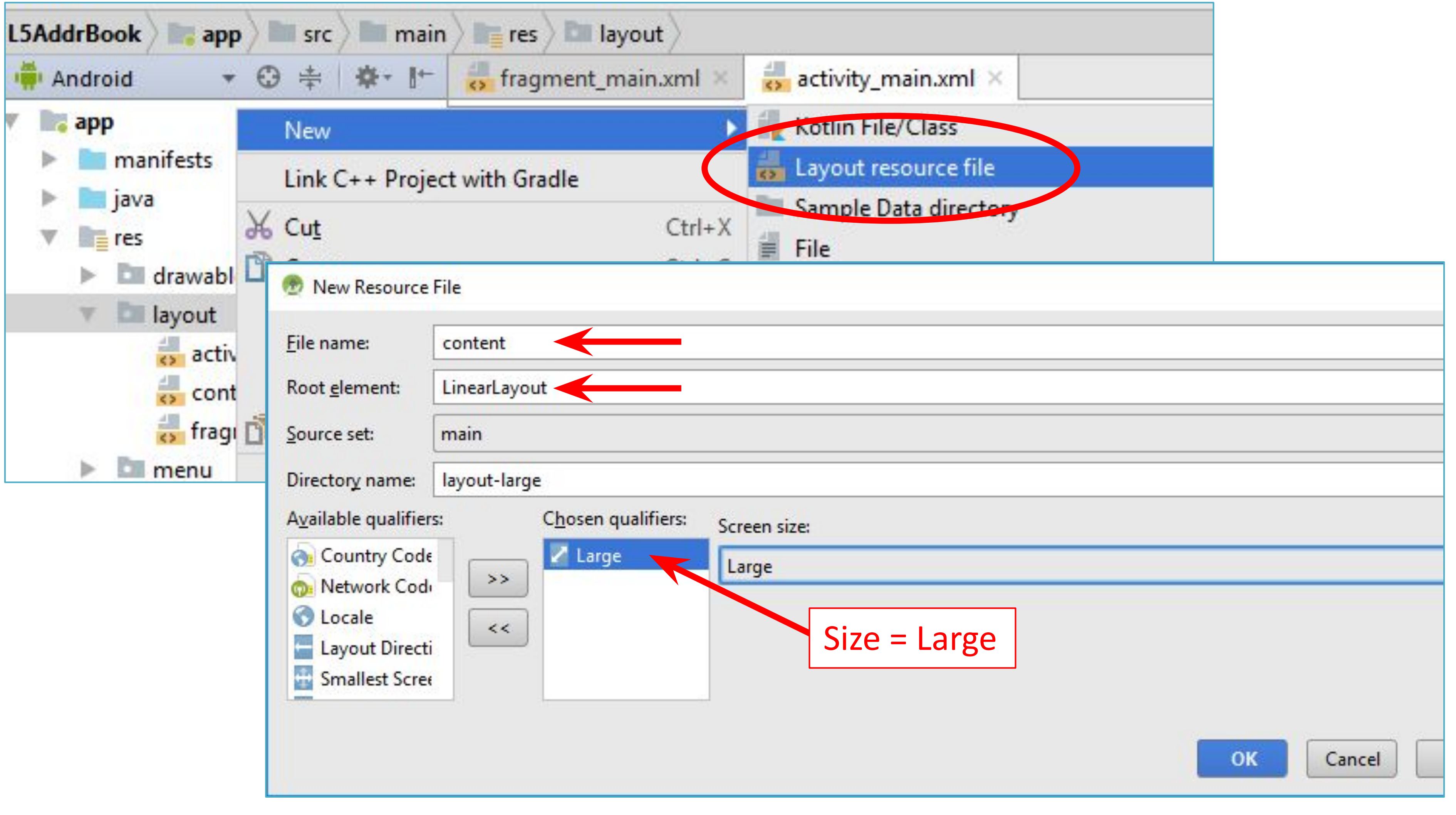
Построение графического интерфейса. Макет для телефона



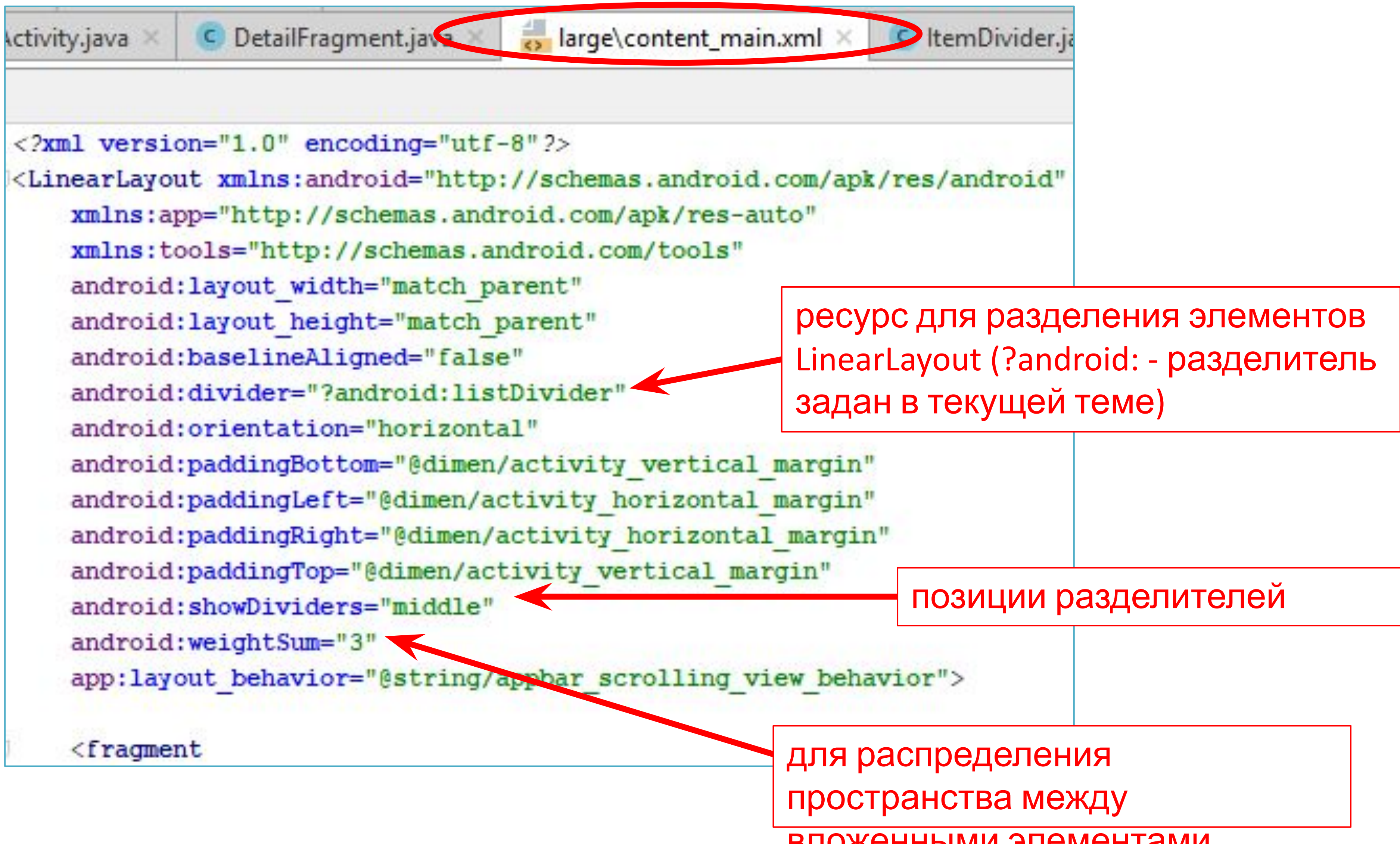
- fragmentContainer используется главной активностью для динамического отображения фрагментов
- свойство app:layout_behavior используется компонентом CoordinatorLayout из activity_main.xml для управления взаимодействиями между представлениями; задание свойства гарантирует, что содержимое FrameLayout будет располагаться под объектом Toolbar, определенным в activity_main.xml

Построение графического интерфейса. Макет для планшета

- список контактов и описание выбранного контакта должны отображаться одновременно



Построение графического интерфейса. Макет для планшета



Построение графического интерфейса. Макет для планшета


```
    app:layout_behavior="@string/appbar_scrolling_view_behavior">

    <fragment
        android:id="@+id/contactsFragment"
        android:name="com.example.someone.15addrbook.ContactsFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_marginEnd="@dimen/divider_margin"
        android:layout_weight="1"
        tools:layout="@layout/fragment_contacts" />

    <FrameLayout
        android:id="@+id/rightPaneContainer"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_marginStart="@dimen/divider_margin"
        android:layout_weight="2" />
</LinearLayout>
```

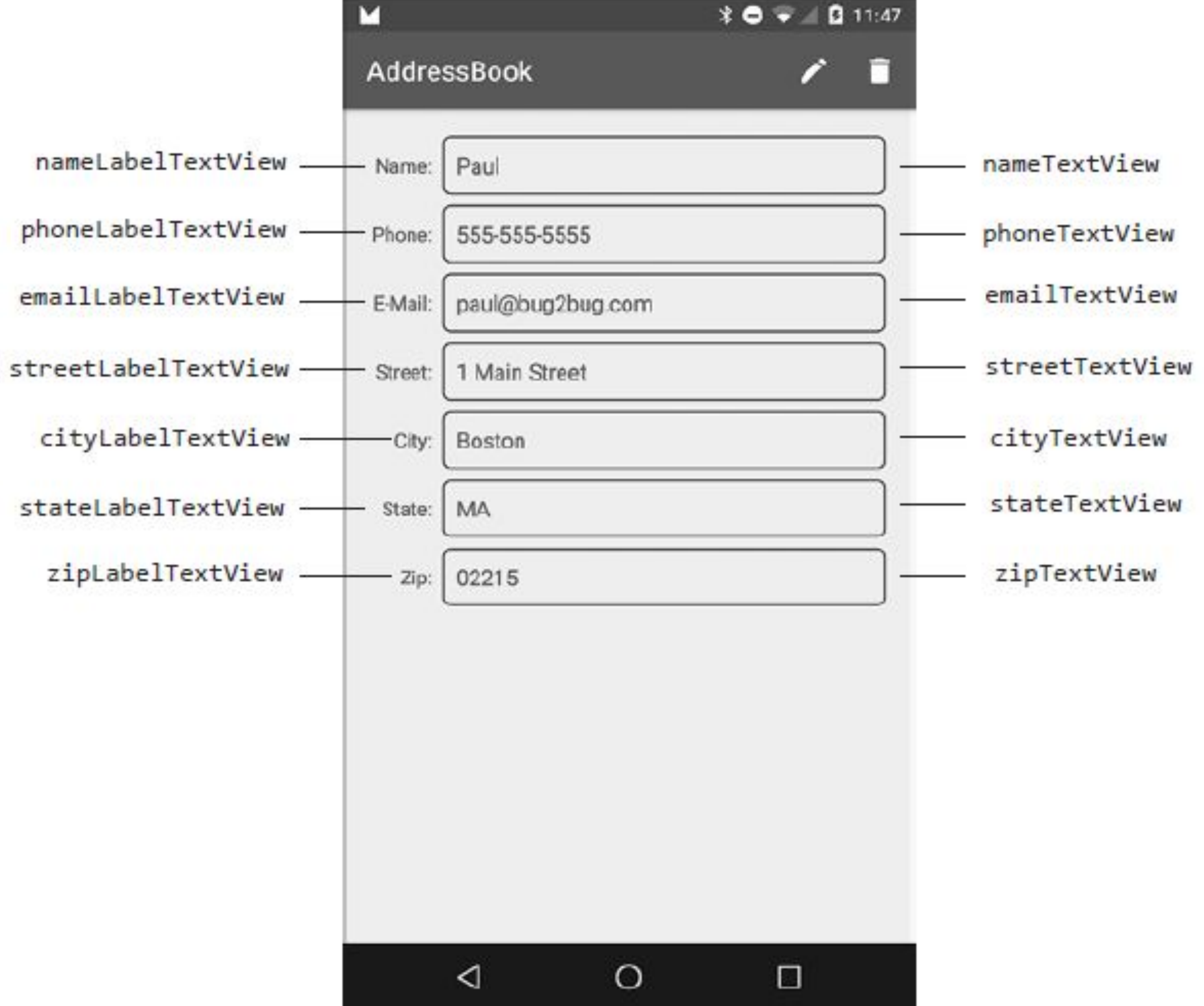
распределение пространства между фрагментом (1/3) и фреймом (2/3)

- определить ресурс @dimen/divider_margin = 16dp
- @layout/fragment_contacts описан далее

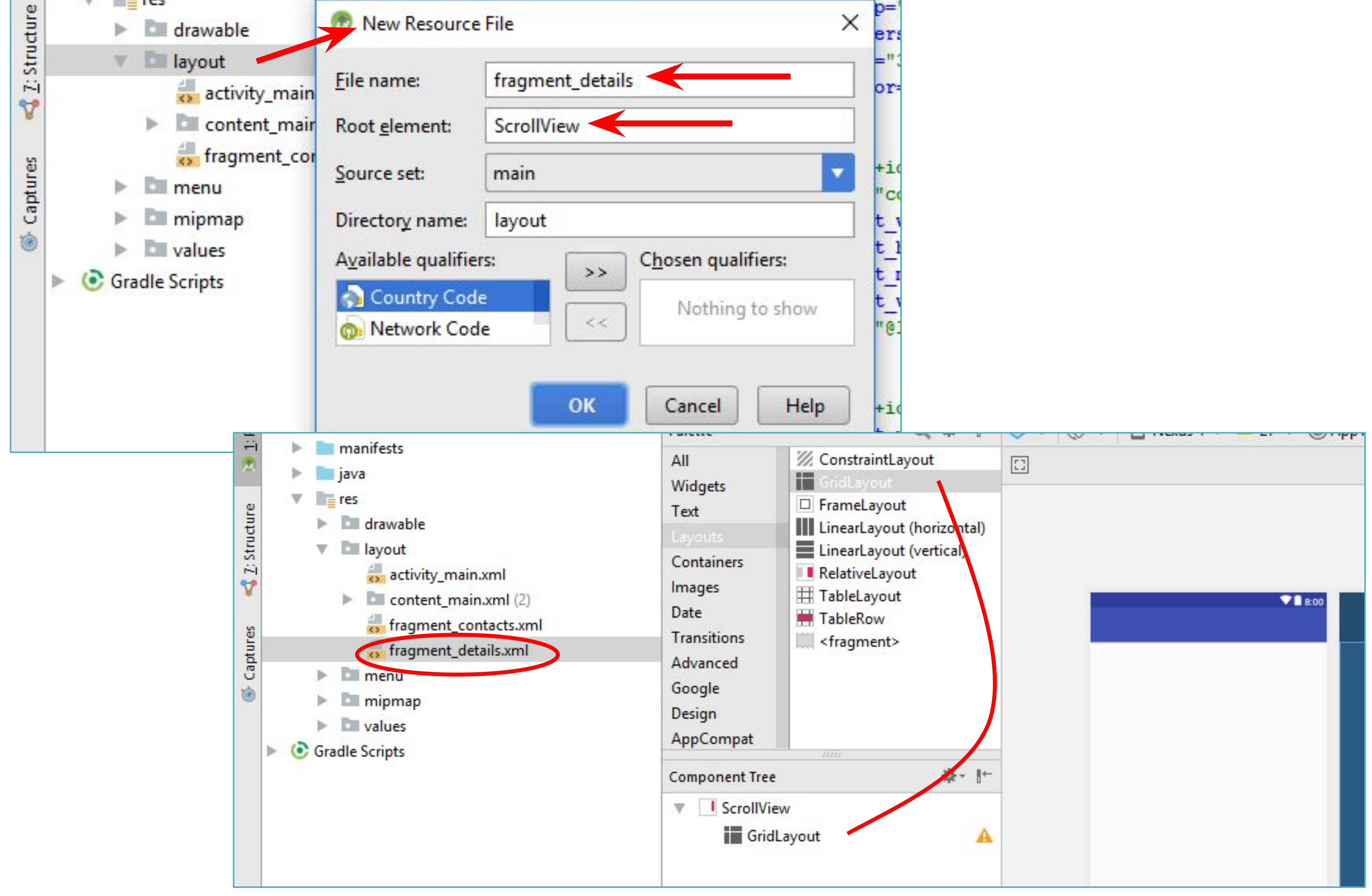
Построение графического интерфейса. Макет ContactsFragment (список контактов)

- переименовать fragment_main.xml в fragment_contacts.xml (рефакторинг)
- удалить TextView
- заменить android.support.constraint.ConstraintLayout на FrameLayout
- оставить в свойствах FrameLayout только:
  - xmlns:android
  - android:layout_width
  - android layout_height
- добавить компонент android.support.v7.widget.**RecyclerView**
  - id="recyclerView"
  - layout_width="match_parent"
  - layout_height="match_parent"
- добавить компонент android.support.design.widget.FloatingActionButton
  - id=addButton
  - layout_width="wrap_content"
  - layout_height="wrap_content"
  - layout_gravity="top|end"
  - layout_margin="@dimen/fab_margin"
  - src="@drawable/ic_add_24dp"

Построение графического интерфейса. Макет DetailFragment (описание контакта)



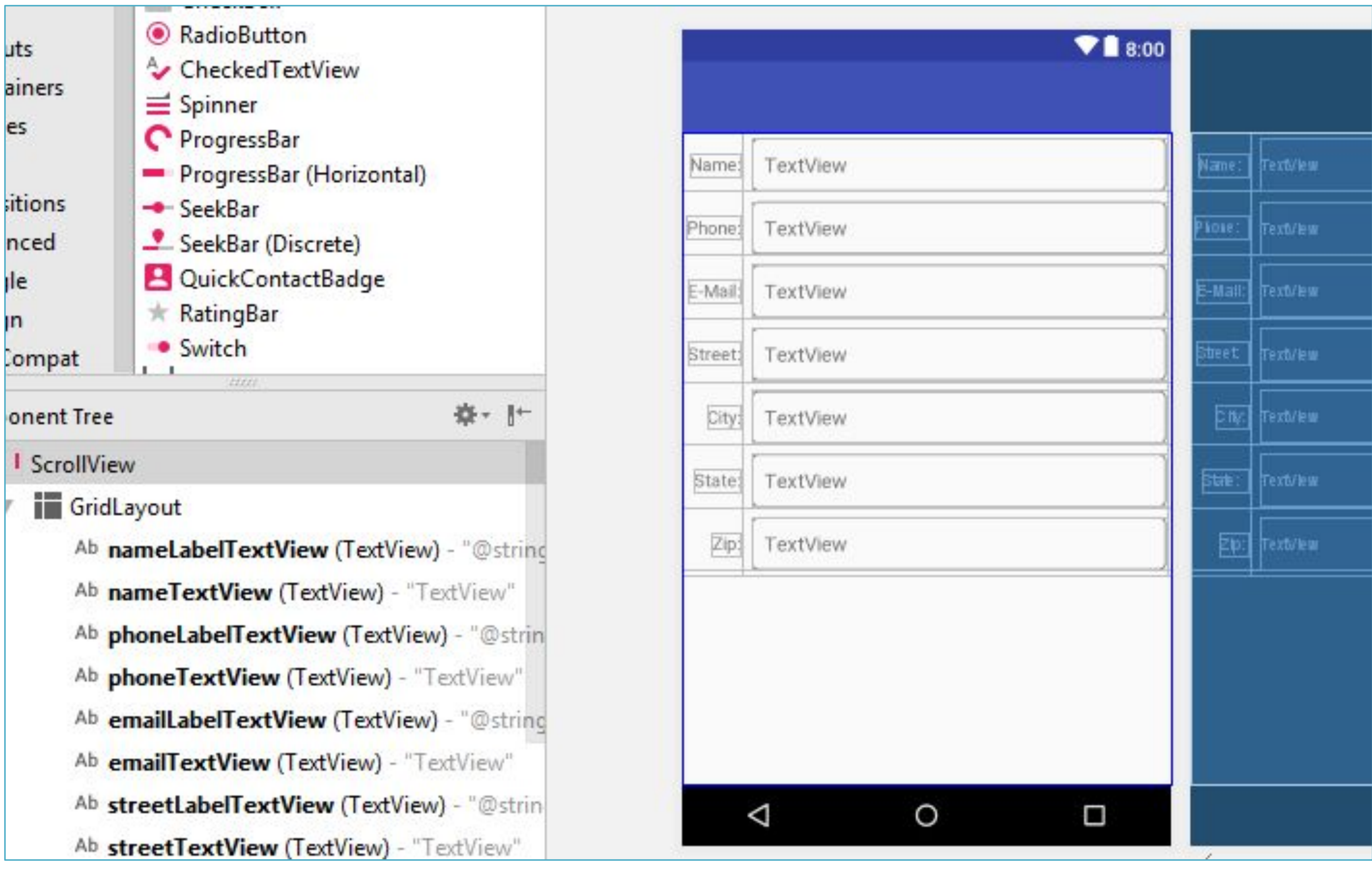
Построение графического интерфейса. Макет DetailFragment (описание контакта)



Построение графического интерфейса. Макет DetailFragment (описание контакта)

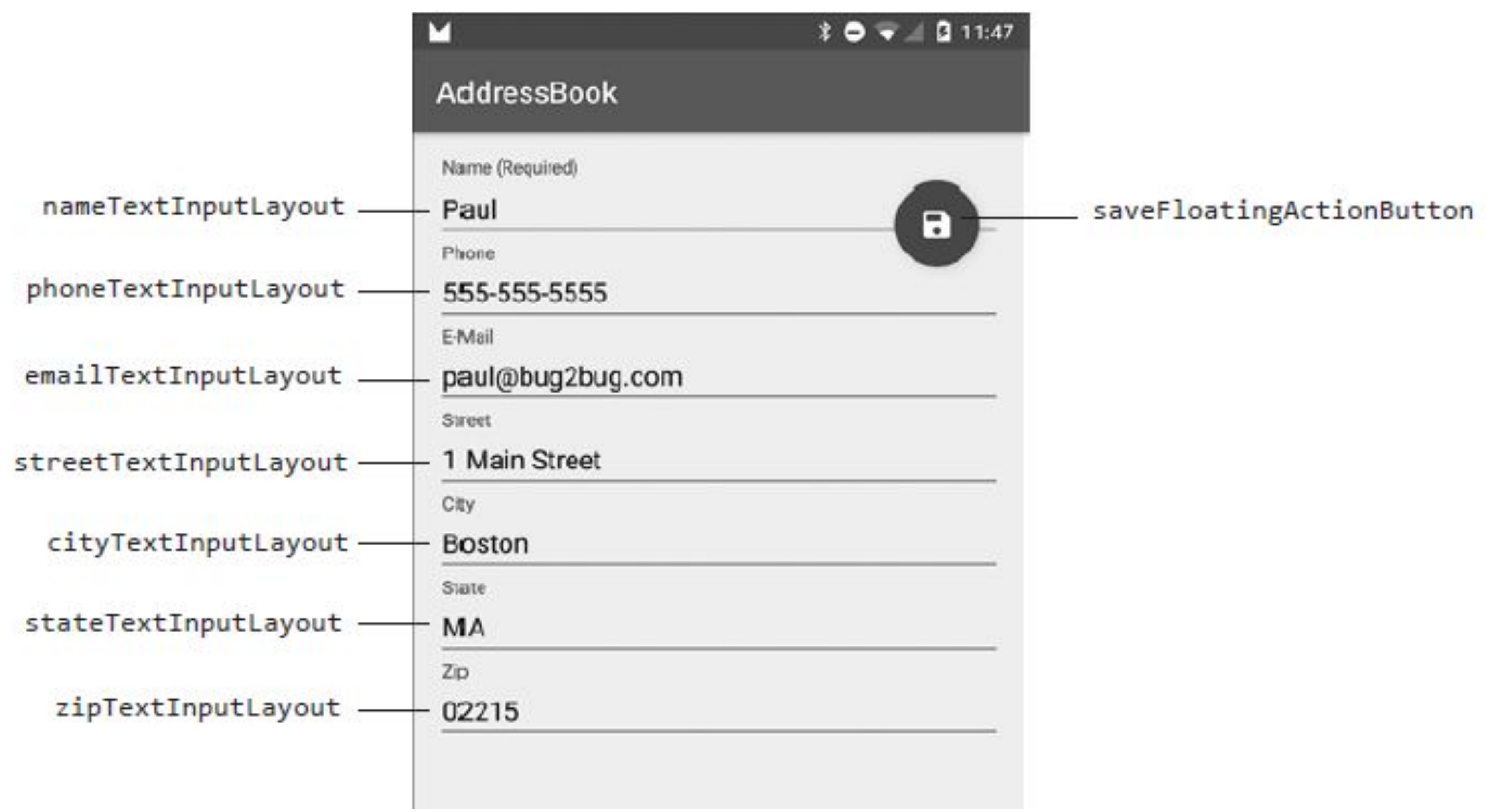
- настраиваем **GridLayout**
  .layout.width="match_parent"
  .layout.height="wrap_content" (чтобы родительский ScrollView определил высоту GridLayout и необходимость прокрутки)
  .columnCount=2
  .useDefaultMargins=true
- настраиваем компоненты **TextView в левом столбце**
  .id - по рисунку макета
  .layout.row=0…6 (в зависимости от строки)
  .layout.column=0
  .text=соответствующий текстовый ресурс
  .style=**@style/ContactLabelTextView**
- настраиваем компоненты **TextView в правом столбце**
  .id - по рисунку макета
  .layout.row=0…6 (в зависимости от строки)
  .layout.column=1
  .style=**@style/ContactTextView**

Построение графического интерфейса. Макет DetailFragment (описание контакта)

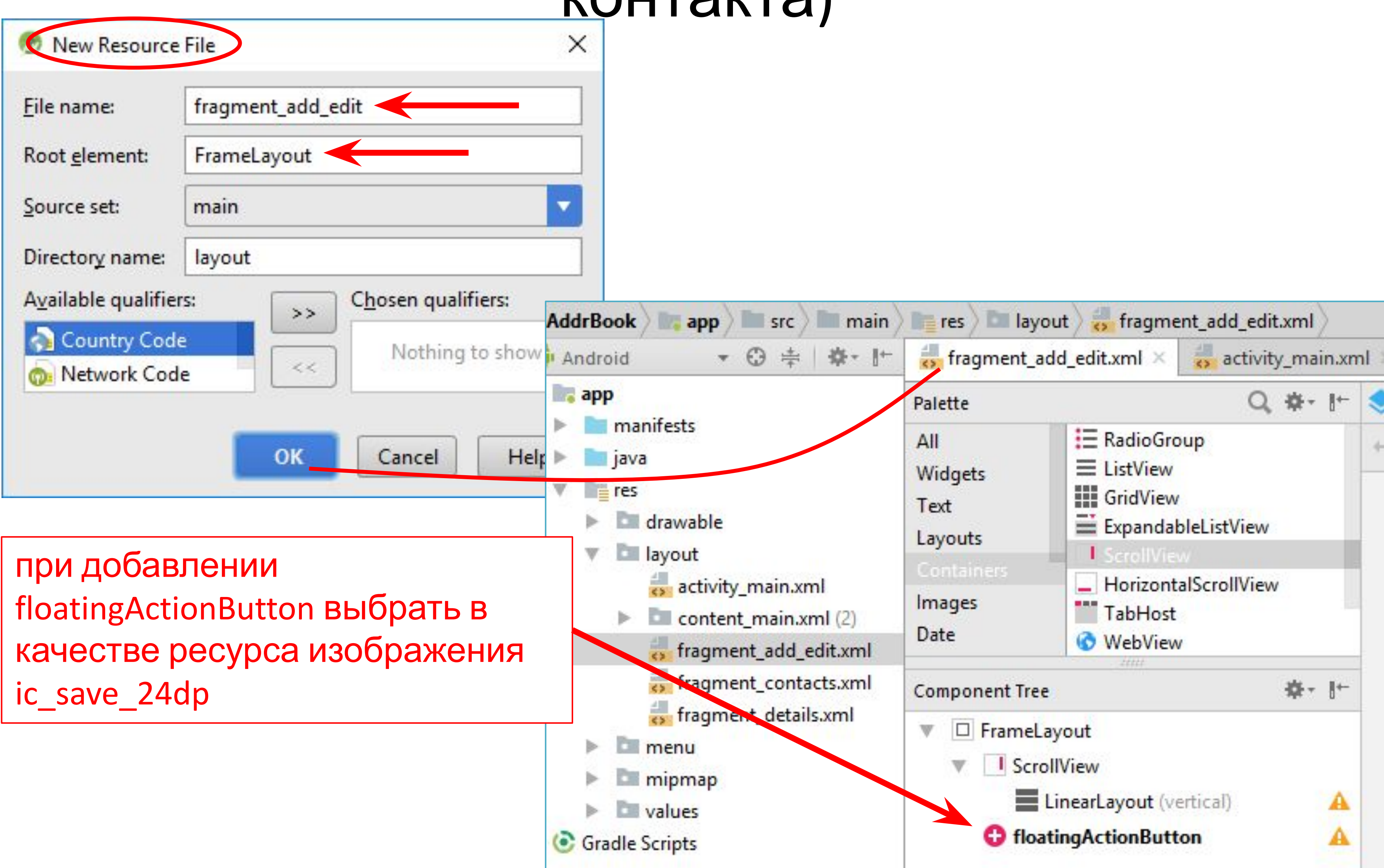


Построение графического интерфейса: Макет AddEditFragment (изменение контакта)

вызывается из ContactsFragment при добавлении или из DetailFragment при редактировании контакта



Построение графического интерфейса. Макет AddEditFragment (изменение контакта)



при добавлении floatingActionButton выбрать в качестве ресурса изображения ic_save_24dp

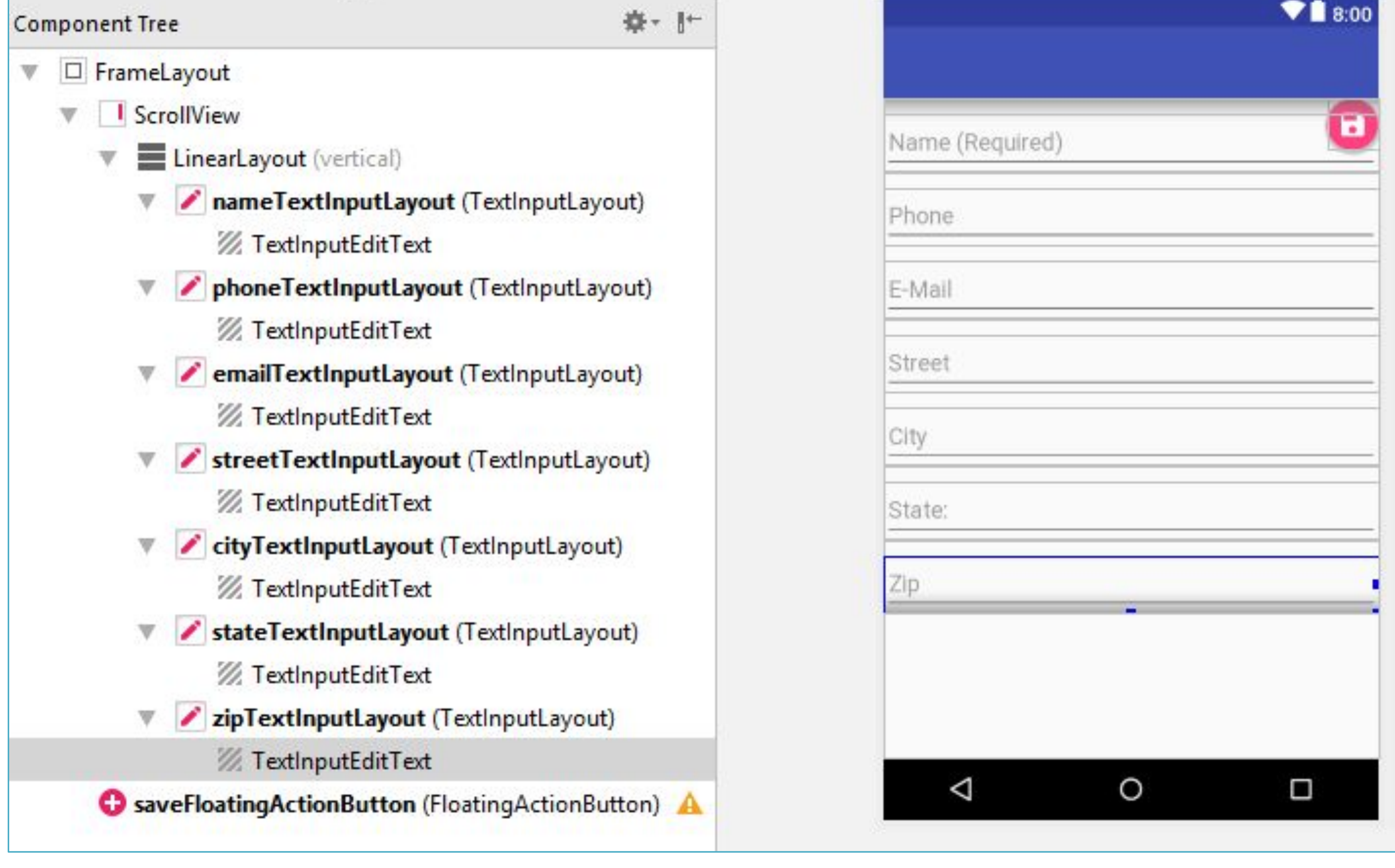
Построение графического интерфейса. Макет AddEditFragment (изменение контакта)

- настраиваем **floatingActionButton**
  .id=saveFloatingActionButton
  .layout_gravity=[top,end]
- настраиваем **scrollView**
  .layout.width=.layout.height=match_parent
- настраиваем **LinearLayout** (при необходимости)
  .layout.width=match_parent
  .layout.height=wrap_content
  .orientation=vertical
- добавляем в LinearLayout семь элементов **TextInputLayout**
- настраиваем элементы **TextInputLayout**
  .id задаём в соответствии с рисунком выше
  .layout.width=match_parent
  .layout.height=wrap_content

Построение графического интерфейса. Макет AddEditFragment (изменение контакта)

- настраиваем **TextInputEditText**
  .hint=@string/hint_... (выбрать соответствующие полям ресурсы)
  .imeOptions=actionDone (в поле zipTextInputLayout, чтобы можно было скрыть экранную клавиатуру)
  .imeOptions=actionNext (для остальных полей, чтобы упростить переход между элементами
  .inputType=… (задаёт тип клавиатуры)
  - в nameTextInputLayout : textPersonName и textCapWords
  - в phoneTextInputLayout: phone
  - в emailTextInputLayout: textEmailAddress
  - в streetTextInputLayout: textPostalAddress и textCapWords
  - в cityTextInputLayout: textPostalAddress и textCapWords
  - в stateTextInputLayout: textPostalAddress и textCapWords
  - в zipTextInputLayout: number

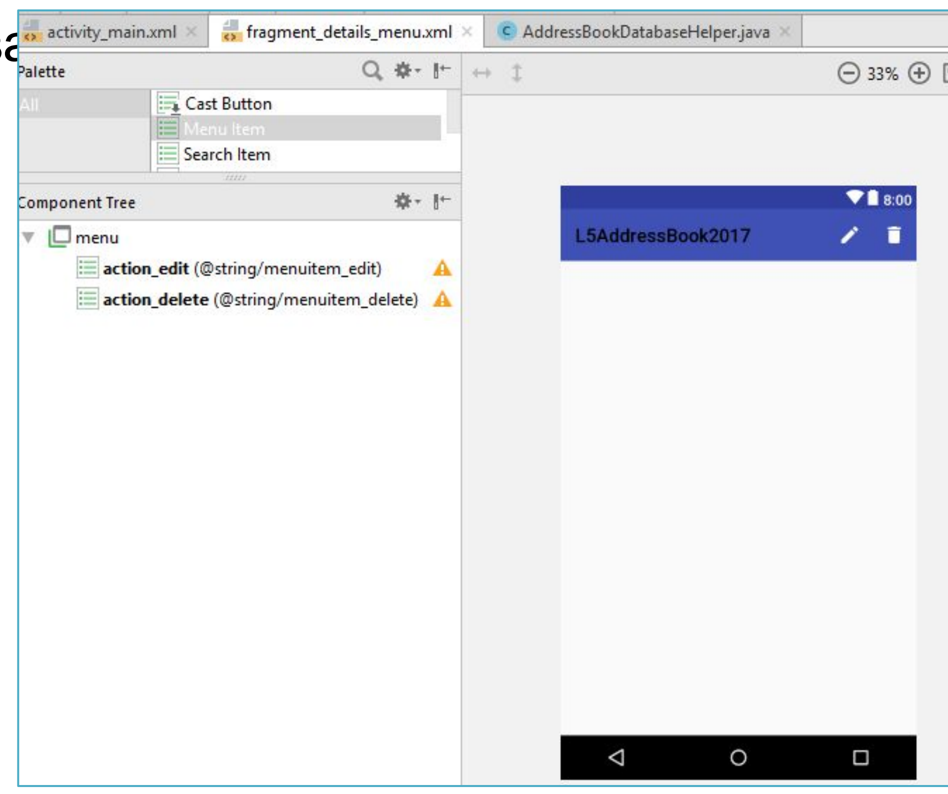
Построение графического интерфейса. Макет AddEditFragment (изменение контакта)



Создание меню

Главной активности меню не нужно. Меню будет использовать DetailFragment для изменения и удаления контактов.

- Удалить методы onCreateOptionsMenu() и onOptionsItemSelected() из класса MainActivity
- Переименовать menu_main.xml в fragment_details_menu.xml
- В редакторе меню удалить пункт Options
- Добавить два экземпляра Menu Item
- Для первого из них задать свойства
  - .id=action_edit
  - .orderInCategory=1
  - .title=@string/menuitem_edit
  - .icon=@drawable/ic_edit_24dp
  - .showAsAction=always
- Для второго задать свойства:
  - .id=action_delete
  - .orderInCategory=2
  - .title=@string/menuitem_delete
  - .icon=@drawable/ic_delete_24dp
  - .showAsAction=always



Описание классов

Пакет data. Классы, относящиеся к работе с БД SQLite.

- **DatabaseDescription** — содержит открытые статические поля, используемые классами ContentProvider и ContentResolver. Вложенный класс **Contact** определяет статические поля для имени таблицы базы данных, Uri для обращения к таблице через ContentProvider, имен столбцов таблицы, а также содержит статический метод для создания объекта Uri, ссылающегося на конкретный контакт в базе данных
- **AddressBookDatabaseHelper** — субкласс SQLiteOpenHelper, который создает базу данных и дает возможность AddressBookContentProvider обращаться к ней
- **AddressBookContentProvider** — субкласс ContentProvider, определяющий операции получения данных, вставки, обновления и удаления с базой данных

Описание классов

«Корневой» пакет. Классы, определяющие главную активность, фрагменты и адаптер приложения, используемые для отображения информации из базы данных в RecyclerView.

- **MainActivity** — управляет фрагментами приложения и реализует их методы интерфейса обратного вызова (выбор контакта, добавление нового, обновление или удаление существующего контакта)
- **ContactsFragment** — управляет списком RecyclerView и кнопкой FloatingActionButton для добавления контактов. Вложенный интерфейс ContactsFragment определяет методы обратного вызова, реализуемые MainActivity, чтобы активность могла реагировать на выбор или добавление контакта
- **ContactsAdapter** — субкласс RecyclerView.Adapter, используемый компонентом RecyclerView фрагмента ContactsFragment для связывания отсортированного списка

Описание классов

«Корневой» пакет (продолжение).

- **AddEditFragment** — управляет компонентами TextInputLayout и кнопкой FloatingActionButton. Вложенный интерфейс AddEditFragment определяет метод обратного вызова, реализуемый MainActivity, чтобы активность могла реагировать на сохранение нового или обновленного контакта
- **DetailFragment** — управляет компонентами TextView с инфор-мацией о выбранном контакте, и командами на панели прило-жения для редактирования или удаления текущего контакта. Вложенный интерфейс DetailFragment определяет методы обратного вызова, реализуемые MainActivity, чтобы активность могла реагировать на удаление контакта или прикосновение к команде на панели приложения для редактирования контакта
- **ItemDivider** — определяет разделитель, отображаемый между элементами компонента RecyclerView фрагмента

Класс DatabaseDescription

Дополнительные библиотеки и статические поля

```
// Класс описывает имя таблицы и имена столбцов базы данных, а также
// содержит другую информацию, необходимую для ContentProvider
package com.example.someone.15addrbook.data;

import android.content.ContentUris;
import android.net.Uri;
import android.provider.BaseColumns;

public class DatabaseDescription {
    // Имя ContentProvider: обычно совпадает с именем пакета
    public static final String AUTHORITY =
            "com.example.someone.15addrbook.data";

    // Базовый URI для взаимодействия с ContentProvider
    private static final Uri BASE_CONTENT_URI =
            Uri.parse("content://" + AUTHORITY);
```

Каждый идентификатор URI, используемый для обращения к конкретному объекту `ContentProvider`, начинается с префикса `"content://"`, за которым следует авторитетное имя — базовый идентификатор URI объекта `ContentProvider`

Класс DatabaseDescription

Вложенный класс с описанием таблицы

```
    // Вложенный класс, определяющий содержимое таблицы contacts
    public static final class Contact implements BaseColumns {
        public static final String TABLE_NAME = "contacts"; // Имя таблицы
        // Объект Uri для таблицы contacts
        public static final Uri CONTENT_URI =
                BASE_CONTENT_URI.buildUpon().appendPath(TABLE_NAME).build();

        // Имена столбцов таблицы
        public static final String COLUMN_NAME = "name";
        public static final String COLUMN_PHONE = "phone";
        public static final String COLUMN_EMAIL = "email";
        public static final String COLUMN_STREET = "street";
        public static final String COLUMN_CITY = "city";
        public static final String COLUMN_STATE = "state";
        public static final String COLUMN_ZIP = "zip";

        // Создание Uri для конкретного контакта
        public static Uri buildContactUri(long id) {
            return ContentUris.withAppendedId(CONTENT_URI, id);
        }
    }
}
```

Класс DatabaseDescription

- Для каждой таблицы базы данных обычно создается класс, сходный с классом Contact
- Имя таблицы и имена столбцов будут использоваться при создании базы данных классом AddressBookDatabaseHelper
- Класс ContentUris (пакет android.content) содержит статические вспомогательные методы для выполнения операций с URI "content://". Метод с withAppendedId присоединяет косую черту (/) и идентификатор записи к объекту Uri в первом аргументе.
- В таблице базы данных каждой строке обычно присваивается первичный ключ, однозначно идентифицирующий строку. При работе с ListView и Cursor этому столбцу должно быть присвоено имя "_id" — Android также использует его для столбца ID в таблицах баз данных SQLite. Для RecyclerView это имя не является обязательным, но оно используется из-за сходства между ListView и RecyclerView и из-за применения Cursor и базы данных SQLite. Вместо того чтобы определять эту константу прямо в классе Contact, он реализует интерфейс BaseColumns (пакет android.provider), определяющий константу _ID со значением "_id".

Класс AddressBookDatabaseHelper

Расширяет абстрактный класс SQLiteOpenHelper, упрощающий создание баз данных и управление изменениями их версий.

```
// Субкласс SQLiteOpenHelper, определяющий базу данных приложения
package com.example.someone.15addrbook.data;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

import com.example.someone.15addrbook.data.DatabaseDescription.Contact;

class AddressBookDatabaseHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "AddressBook.db";
    private static final int DATABASE_VERSION = 1;

    // Конструктор
    public AddressBookDatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
```

- в конструкторе вызывается конструктор родительского класса
- его аргументы: объект `Context`, в котором создается или открывается база данных; имя базы данных — может быть равно `null`, если база данных существует только в памяти; объект `CursorFactory` — `null` означает, что используется объект `CursorFactory` по умолчанию; номер версии базы данных

Класс AddressBookDatabaseHelper

```
// Создание таблицы contacts при создании базы данных
@Override
public void onCreate(SQLiteDatabase db) {
    // Команда SQL для создания таблицы contacts
    final String CREATE_CONTACTS_TABLE =
            "CREATE TABLE " + Contact.TABLE_NAME + "(" +
                    Contact._ID + " integer primary key, " +
                    Contact.COLUMN_NAME + " TEXT, " +
                    Contact.COLUMN_PHONE + " TEXT, " +
                    Contact.COLUMN_EMAIL + " TEXT, " +
                    Contact.COLUMN_STREET + " TEXT, " +
                    Contact.COLUMN_CITY + " TEXT, " +
                    Contact.COLUMN_STATE + " TEXT, " +
                    Contact.COLUMN_ZIP + " TEXT);";
    db.execSQL(CREATE_CONTACTS_TABLE); // Создание таблицы contacts
}

// Обычно определяет способ обновления при изменении схемы базы данн
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion,
                      int newVersion) {
}
}
```

onCreate() вызывается, если база данных не существует

при повышении или понижении версии БД вызываются методы **onUpgrade()** или **onDowngrade()** (здесь они не используются)

Класс AddressBookContentProvider

Определяет, как должны выполняться операции `query`, `insert`, `update` и `delete` с базой данных.

Дополнительные библиотеки

```
// Субкласс ContentProvider для работы с базой данных приложения
package com.example.someone.l5addrbook.data;

import android.content.ContentProvider;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;

import com.example.someone.l5addrbook.R;
import com.example.someone.l5addrbook.data.DatabaseDescription.Contact;
```

Класс AddressBookContentProvider

Поля класса

```
public class AddressBookContentProvider extends ContentProvider {
    // Используется для обращения к базе данных
    private AddressBookDatabaseHelper dbHelper;

    // UriMatcher помогает ContentProvider определить выполняемую операцию
    private static final UriMatcher uriMatcher =
            new UriMatcher(UriMatcher.NO_MATCH);

    // Константы, используемые для определения выполняемой операции
    private static final int ONE_CONTACT = 1; // Один контакт
    private static final int CONTACTS = 2; // Таблица контактов
```

- **dbHelper** — ссылка на объект AddressBookDatabaseHelper, который создает базу данных и разрешает объекту ContentProvider выполнять с базой данных операции чтения и записи
- **uriMatcher**— статическая переменная, содержащая объект класса UriMatcher (пакет android.content). ContentProvider использует UriMatcher для определения того, какие операции должны выполняться в методах query, insert, update и delete
- **UriMatcher** возвращает целочисленные константы ONE_CONTACT и CONTACTS — ContentProvider использует их в командах switch в методах query, insert, update и delete

Класс AddressBookContentProvider

Поля класса

```
// Статический блок для настройки UriMatcher объекта ContentProvider
static {
    // Uri для контакта с заданным идентификатором
    uriMatcher.addURI(DatabaseDescription.AUTHORITY,
            Contact.TABLE_NAME + "/#", ONE_CONTACT);

    // Uri для таблицы
    uriMatcher.addURI(DatabaseDescription.AUTHORITY,
            Contact.TABLE_NAME, CONTACTS);
}
```

- Статический блок добавляет объекты Uri в статический объект UriMatcher. Этот блок выполняется один раз, когда класс AddressBookContentProvider загружается в память.
- Сначала добавляется URI в форме *content://com.example.someone.l5addrbook.data/contacts/#,* где # — метасимвол, обозначающий последовательность числовых символов — в данном случае уникальное значение первичного ключа для одного контакта в таблице contacts. Когда URI соответствует этому формату, UriMatcher возвращает константу ONE_CONTACT.
- Затем добавляется URI в форме *content://com. example.someone.l5addrbook.data/contacts,* представляющий всю таблицу contacts. Когда URI соответствует этому формату, UriMatcher возвращает константу CONTACTS.

Класс AddressBookContentProvider

```
// Вызывается при создании AddressBookContentProvider
@Override
public boolean onCreate() {
    // Создание объекта AddressBookDatabaseHelper
    dbHelper = new AddressBookDatabaseHelper(getContext());
    return true; // Объект ContentProvider создан успешно
}

// Обязательный метод: здесь не используется, возвращаем null
@Override
public String getType(Uri uri) {
    return null;
}
```

- ContentProvider создаётся при получении первого запроса от ContentResolver
- AddressBookDatabaseHelper используется провайдером для обращения к базе данных
- getType() — обязательный метод ContentProvider, который в данном примере просто возвращает null. Метод обычно используется при создании и запуске интентов для URI с конкретными типами MIME. На основании типов MIME Android может выбирать активности для обработки интентов

Класс AddressBookContentProvider

```
// Получение информации из базы данных
@Override
public Cursor query(Uri uri, String[] projection,
                    String selection, String[] selectionArgs, String sortOrder) {
```

query() получает данные из источника данных провайдера — в данном случае базы данных. Он возвращает объект Cursor, используемый для работы с результатами.

Аргументы:

- uri — объект Uri, представляющий загружаемые данные.
- projection — столбцы, которые должен вернуть запрос. Если аргумент равен null, то в результат включаются все столбцы.
- selection — условие SQL WHERE, заданное без ключевого слова WHERE. Если этот аргумент равен null, то все строки будут включены в результат.
- selectionArgs — строки, заменяющие все заполнители аргументов (?) в строке selection.
- sortOrder — условие SQL ORDER BY, заданное без ключевых слов ORDER BY. Если этот аргумент равен null, то порядок сортировки определяется провайдером — таким образом, без явного указания порядка сортировки последовательность возвращаемых результатов не гарантирована.

Класс AddressBookContentProvider

```
// Создание SQLiteQueryBuilder для запроса к таблице contacts
SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
queryBuilder.setTables(Contact.TABLE_NAME);
```

- **queryBuilder** - объект SQLiteQueryBuilder (пакет android.database.sqlite) для построения запросов SQL, передаваемых базе данных SQLite
- метод setTables() указывает, что запрос будет выбирать данные из таблицы contacts базы данных. Строковый аргумент этого метода может использоваться для выполнения операций соединений таблиц; для этого таблицы перечисляются через запятую или используется синтаксис условия SQL JOIN.

Класс AddressBookContentProvider

```
switch (uriMatcher.match(uri)) {
    case ONE_CONTACT: // Выбрать контакт с заданным идентификатором
        queryBuilder.appendWhere(
                Contact._ID + "=" + uri.getLastPathSegment());
        break;
    case CONTACTS: // Выбрать все контакты
        break;
    default:
        throw new UnsupportedOperationException(
                getContext().getString(R.string.invalid_query_uri) + uri);
}
```

- Метод match() возвращает одну из констант, зарегистрированных с UriMatcher.
- Если возвращается константа ONE_CONTACT, выбирается только контакт с идентификатором, заданным в Uri. Метод appendWhere() используется для добавления условия WHERE с идентификатором контакта. Метод getLastPathSegment() класса Uri возвращает последний сегмент URI — например, идентификатор контакта 5 в следующем URI: *content://com.example.someone.l5addrbook.data/contacts/5*
- Если возвращается константа CONTACTS, то конструкция switch завершается, не добавляя ничего к запросу — в этом случае будут выбраны все контакты
- Для всех URI, не соответствующих схеме, инициируется исключение UnsupportedOperationException, указывающее на недействительность URI

Класс AddressBookContentProvider

```
// Выполнить запрос для получения одного или всех контактов
Cursor cursor = queryBuilder.query(dbHelper.getReadableDatabase(),
        projection, selection, selectionArgs, null, null, sortOrder);
```

Метод **query()** класса SQLiteQueryBuilder используется для выполнения запроса к базе данных и получения объекта Cursor, представляющего результаты.

Аргументы похожи на аргументы метода ContentProvider.query().

- SQLiteDatabase — база данных, к которой обращен запрос.
- projection — столбцы, которые должен вернуть запрос. Если аргумент равен null, то в результат включаются все столбцы.
- selection —условие SQL WHERE. Если null, то все записи будут включены в результат.
- selectionArgs — аргументами, заменяющие все заполнители аргументов (?) в строке selection.
- groupBy —условие SQL GROUP BY. Если null, то группировка не выполняется.
- having —условие SQL HAVING. Если null, то в результат включаются все группы, заданные аргументом groupBy.
- sortOrder —условие SQL ORDER BY. Если null, то порядок сортировки определяется провайдером — таким образом, без явного указания порядка сортировки последовательность возвращаемых результатов не гарантирована.

Класс AddressBookContentProvider

```
    // Настройка отслеживания изменений в контенте
    cursor.setNotificationUri(getContext().getContentResolver(), uri);
    return cursor;
}
```

- Метод **setNotificationUri()** класса Cursor сообщает, что объект Cursor должен обновляться при изменении данных, на которые он ссылается.
- В первом аргументе передается объект ContentResolver, обращающийся к ContentProvider, а во втором — объект Uri, используемый для обращения.

Класс AddressBookContentProvider

```
// Вставка нового контакта в базу данных
@Override
public Uri insert(Uri uri, ContentValues values) {
    Uri newContactUri = null;
    switch (uriMatcher.match(uri)) {
        case CONTACTS:
            // При успехе возвращается идентификатор записи нового контакта
            long rowId = dbHelper.getWritableDatabase().insert(
                    Contact.TABLE_NAME, null, values);
            // Если контакт был вставлен, создать подходящий Uri;
            // в противном случае выдать исключение
            if (rowId > 0) { // SQLite row IDs start at 1
                newContactUri = Contact.buildContactUri(rowId);
                // Оповестить наблюдателей об изменениях в базе данных
                getContext().getContentResolver().notifyChange(uri, null);
            } else
                throw new SQLException(
                        getContext().getString(R.string.insert_failed) + uri);
            break;
        default:
            throw new UnsupportedOperationException(
                    getContext().getString(R.string.invalid_insert_uri) + uri);
    }
    return newContactUri;
}
```

Класс AddressBookContentProvider

```java
// Вставка нового контакта в базу данных
@Override
public Uri insert(Uri uri, ContentValues values) {
    Uri newContactUri = null;
```

Метод **insert()** добавляет новую запись в таблицу contacts.
Аргументы:

- uri — объект Uri, представляющий таблицу, в которую будут вставлены данные
- values — объект ContentValues с парами «ключ—значение». Имена столбцов являются ключами, а данные, сохраняемые в столбцах, — значениями

```java
switch (uriMatcher.match(uri)) {
    case CONTACTS:
        // При успехе возвращается идентификатор записи нового контакта
        long rowId = dbHelper.getWritableDatabase().insert(
                Contact.TABLE_NAME, null, values);
```

- Выполняется проверка, относится ли URI к таблице contacts. Если URI подходит, то новый контакт добавляется в базу данных.
- Метод getWritableDatabase() класса AddressBookDatabaseHelper предоставляет объект SQLiteDatabaseObject для изменения данных в базе.
- Метод insert() класса SQLiteDatabase вставляет значения из объекта values в таблицу contacts. Второй параметр этого метода не используется в приложении.

Класс AddressBookContentProvider

```
// Если контакт был вставлен, создать подходящий Uri;
// в противном случае выдать исключение
if (rowId > 0) { // SQLite row IDs start at 1
    newContactUri = Contact.buildContactUri(rowId);
    // Оповестить наблюдателей об изменениях в базе данных
    getContext().getContentResolver().notifyChange(uri, null);
} else
    throw new SQLException(
            getContext().getString(R.string.insert_failed) + uri);
break;
```

- Метод SQLiteDatabase.insert() возвращает уникальный идентификатор нового контакта, если вставка завершилась успешно, или –1 в случае неудачи.
- Если вставка успешна (rowID больше 0; в SQLite записи индексируются с 1), то создается Uri для представления нового контакта, а наблюдатель ContentResolver оповещается об изменении базы данных, чтобы клиентский код ContentResolver мог отреагировать на изменения.
- Если rowID не больше 0, то попытка выполнения операции завершается неудачей и инициируется исключение SQLException.

Класс AddressBookContentProvider

```
        default:
            throw new UnsupportedOperationException(
                    getContext().getString(R.string.invalid_insert_uri) + uri);
    }
    return newContactUri;
}
```

- Если URI таблицы недействителен для операции insert, то инициируется исключение UnsupportedOperationException.
- Метод возвращает URI новой записи

Класс AddressBookContentProvider

```
// Обновление существующего контакта в базе данных
@Override
public int update(Uri uri, ContentValues values,
                  String selection, String[] selectionArgs) {
    int numberOfRowsUpdated; // 1, если обновление успешно; 0 при неудаче
    switch (uriMatcher.match(uri)) {
        case ONE_CONTACT:
            // Получение идентификатора контакта из Uri
            String id = uri.getLastPathSegment();
            // Обновление контакта
            numberOfRowsUpdated = dbHelper.getWritableDatabase().update(
                    Contact.TABLE_NAME, values, Contact._ID + "=" + id,
                    selectionArgs);
            break;
        default:
            throw new UnsupportedOperationException(
                    getContext().getString(R.string.invalid_update_uri) + uri);
    }
    // Если были внесены изменения, оповестить наблюдателей
    if (numberOfRowsUpdated != 0) {
        getContext().getContentResolver().notifyChange(uri, null);
    }
    return numberOfRowsUpdated;
}
```

Класс AddressBookContentProvider

```
// Обновление существующего контакта в базе данных
@Override
public int update(Uri uri, ContentValues values,
                  String selection, String[] selectionArgs) {
```

- uri — объект Uri, представляющий таблицу, в которой обновляются данные
- values — объект ContentValues с именами обновляемых столбцов и их значениями
- selection — строка с критериями выборки: условие SQL WHERE, заданное *без* ключевого слова WHERE. Если этот аргумент равен null, то все записи будут включены в результат
- selectionArgs — массив String с аргументами, заменяющими все заполнители аргументов (?) в строке selection

Класс AddressBookContentProvider

```
int numberOfRowsUpdated; // 1, если обновление успешно; 0 при неудаче
switch (uriMatcher.match(uri)) {
    case ONE_CONTACT:
        // Получение идентификатора контакта из Uri
        String id = uri.getLastPathSegment();
        // Обновление контакта
        numberOfRowsUpdated = dbHelper.getWritableDatabase().update(
                Contact.TABLE_NAME, values, Contact._ID + "=" + id,
                selectionArgs);
        break;
```

- Обновление производится только для одного контакта, поэтому выполняется проверка URI на соответствие ONE_CONTACT
- getLastPathSegment() выделяет последнюю часть URI – уникальный идентификатор
- При успешном обновлении SQLiteDatabase.update() вернёт число обновлённых записей (1). Аргументы метода:
  - имя обновляемой таблицы
  - объект ContentValues с именами обновляемых столбцов и их новыми значениями
  - условие SQL WHERE, определяющее обновляемые записи
  - массив String с аргументами, которые подставляются на место заполнителей ? в условии WHERE

Класс AddressBookContentProvider

```
        default:
            throw new UnsupportedOperationException(
                    getContext().getString(R.string.invalid_update_uri) + uri);
    }
    // Если были внесены изменения, оповестить наблюдателей
    if (numberOfRowsUpdated != 0) {
        getContext().getContentResolver().notifyChange(uri, null);
    }
    return numberOfRowsUpdated;
}
```

- В случае попытки некорректной операции генерируется исключение
- Если изменения успешны, то наблюдатель ContentResolver оповещается об изменении базы данных, чтобы клиентский код ContentResolver мог отреагировать на изменения
- Метод возвращает число успешно обновлённых записей

Класс AddressBookContentProvider

```
// Удаление существующего контакта из базы данных
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    int numberOfRowsDeleted;
    switch (uriMatcher.match(uri)) {
        case ONE_CONTACT:
            // Получение из URI идентификатора контакта
            String id = uri.getLastPathSegment();
            // Удаление контакта
            numberOfRowsDeleted = dbHelper.getWritableDatabase().delete(
                    Contact.TABLE_NAME, Contact._ID + "=" + id, selectionArgs);
            break;
        default:
            throw new UnsupportedOperationException(
                    getContext().getString(R.string.invalid_delete_uri) + uri);
    }
    // Оповестить наблюдателей об изменениях в базе данных
    if (numberOfRowsDeleted != 0) {
        getContext().getContentResolver().notifyChange(uri, null);
    }
    return numberOfRowsDeleted;
}

}
```

Класс AddressBookContentProvider

```java
// Удаление существующего контакта из базы данных
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    int numberOfRowsDeleted;
    switch (uriMatcher.match(uri)) {
        case ONE_CONTACT:
            // Получение из URI идентификатора контакта
            String id = uri.getLastPathSegment();
            // Удаление контакта
            numberOfRowsDeleted = dbHelper.getWritableDatabase().delete(
                    Contact.TABLE_NAME, Contact._ID + "=" + id, selectionArgs);
            break;
```

- Аргументы AddressBookContentProvider:
  - uri — объект Uri, представляющий таблицу, в которой обновляются данные
  - selection — строка с условием SQL WHERE, определяющим удаляемые записи
  - selectionArgs — массив String с аргументами, заменяющими все заполнители аргументов (?) в строке selection
- Удаление производится только для одного контакта, поэтому выполняется проверка URI на соответствие ONE_CONTACT
- При успешном удалении SQLiteDatabase.delete() вернёт число удалённых записей (1).

Класс AddressBookContentProvider

```
            default:
                throw new UnsupportedOperationException(
                        getContext().getString(R.string.invalid_delete_uri) + uri);
        }
        // Оповестить наблюдателей об изменениях в базе данных
        if (numberOfRowsDeleted != 0) {
            getContext().getContentResolver().notifyChange(uri, null);
        }
        return numberOfRowsDeleted;
    }

}
```

- В случае попытки некорректной операции генерируется исключение
- Если изменения успешны, то наблюдатель ContentResolver оповещается об изменении базы данных, чтобы клиентский код ContentResolver мог отреагировать на изменения
- Метод возвращает число успешно удалённых записей

Класс MainActivity

Управляет фрагментами приложения и координирует взаимодействия между ними.

На телефонах MainActivity в любой момент времени отображает только один фрагмент начиная с ContactsFragment. На планшетах MainActivity всегда отображает ContactsFragment слева и в зависимости от контекста — либо DetailFragment, либо AddEditFragment в правых 2/3 экрана.

Дополнительные библиотеки

```
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import java.io.Serializable;
```

Класс `FragmentTransaction` из библиотеки поддержки v4 используется главной активностью для добавления и удаления фрагментов приложения.

Класс MainActivity

Суперкласс, реализуемые интерфейсы и поля

```
public class MainActivity extends AppCompatActivity
        implements ContactsFragment.ContactsFragment,
        DetailFragment.DetailFragmentListener,
        AddEditFragment.AddEditFragmentListener,
        Serializable {

    // Ключ для сохранения Uri контакта в переданном объекте Bundle
    public static final String CONTACT_URI = "contact_uri";
    private ContactsFragment contactsFragment; // Вывод списка контактов
```

интерфейсы станут доступны после описания соответствующих классов

`MainActivity` реализует четыре интерфейса.

- **`ContactsFragment.ContactsFragment`** содержит методы, при помощи которых `ContactsFragment` сообщает `MainActivity`, что пользователь выбрал контакт в списке или добавил новый контакт
- **`DetailFragment.DetailFragmentListener`** содержит методы, при помощи которых `DetailFragment` сообщает `MainActivity`, что пользователь удаляет или хочет отредактировать существующий контакт.
- **`AddEditFragment.AddEditFragmentListener`** содержит методы, при помощи которых `AddEditFragment` сообщает `MainActivity`, что пользователь завершил добавление нового контакта или редактирование существующего контакта.
- **Serializable** используется для передачи ссылки на главную активность через объект **Bundle** в класс диалога подтверждения удаления контакта в классе

Класс MainActivity

Переопределение onCreate()

```
// Отображает ContactsFragment при первой загрузке MainActivity
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
```

- Метод onCreate() заполняет графический интерфейс MainActivity.
- Если приложение выполняется на телефоне, метод отображает ContactsFragment.
- Если активность восстанавливается после завершения или создается повторно после изменения конфигурации, значение savedInstanceState будет отлично от null.

Класс MainActivity

Переопределение onCreate()

```
    // Если макет содержит fragmentContainer, используется макет
    // для телефона; отобразить ContactsFragment
    if (savedInstanceState == null &&
            findViewById(R.id.fragmentContainer) != null) {
        // Создание ContactsFragment
        contactsFragment = new ContactsFragment();

        // Добавление фрагмента в FrameLayout
        FragmentTransaction transaction =
                getSupportFragmentManager().beginTransaction();
        transaction.add(R.id.fragmentContainer, contactsFragment);
        transaction.commit(); // Вывод ContactsFragment
    } else {
        contactsFragment =
                (ContactsFragment) getSupportFragmentManager().
                        findFragmentById(R.id.contactsFragment);
    }
}
```

- FragmentTransaction используется для добавления фрагмента в пользовательский интерфейс.
- Метод FragmentTransaction.add() указывает, что при завершении FragmentTransaction фрагмент ContactsFragment должен быть присоединен к представлению с идентификатором, передаваемым в

Класс MainActivity

Методы ContactsFragment.ContactsFragment

```
// Отображение DetailFragment для выбранного контакта
@Override
public void onContactSelected(Uri contactUri) {
    if (findViewById(R.id.fragmentContainer) != null) // Телефон
        displayContact(contactUri, R.id.fragmentContainer);
    else { // Планшет
        // Извлечение с вершины стека возврата
        getSupportFragmentManager().popBackStack();
        displayContact(contactUri, R.id.rightPaneContainer);
    }
}
```

- вызывается объектом ContactsFragment для оповещения MainActivity о том, что пользователь выбрал контакт для отображения
- для телефона происходит замена фрагмента ContactsFragment на DetailFrragment
- для планшета происходит извлечение верхнего фрагмента из стека возврата и замена содержимого rightPaneContainer фрагментом DetailFragment

Класс MainActivity

Методы ContactsFragment.ContactsFragment

```
// Отображение AddEditFragment для добавления нового контакта
@Override
public void onAddContact() {
    if (findViewById(R.id.fragmentContainer) != null) // Телефон
        displayAddEditFragment(R.id.fragmentContainer, null);
    else // Планшет
        displayAddEditFragment(R.id.rightPaneContainer, null);
}
```

- вызывается объектом ContactsFragment для оповещения MainActivity о том, что пользователь выбрал команду добавления нового контакта
- для телефона AddEditFragment отображается в элементе fragmentContainer
- для планшета AddEditFragment отображается в rightPaneContainer
- передача null в displayAddEditFragment() означает, что добавляется новый контакт; в противном случае объект Bundle (второй аргумент) включает Uri существующего контакта.

Класс MainActivity

Метод для отображения контакта

```
// Отображение информации о контакте
private void displayContact(Uri contactUri, int viewID) {
    DetailFragment detailFragment = new DetailFragment();

    // Передача URI контакта в аргументе DetailFragment
    Bundle arguments = new Bundle();
    arguments.putParcelable(CONTACT_URI, contactUri);
    detailFragment.setArguments(arguments);

    // Использование FragmentTransaction для отображения
    FragmentTransaction transaction =
            getSupportFragmentManager().beginTransaction();
    transaction.replace(viewID, detailFragment);
    transaction.addToBackStack(null);
    transaction.commit(); // Приводит к отображению DetailFragment
}
```

- для передачи URI выбранного контакта в DetailFragment используется объект Bundle (пара «ключ-значение»)
- transaction.replace() указывает, что при завершении FragmentTransaction фрагмент DetailFragment должен заменить содержимое представления с идентификатором, переданным в первом аргументе
- DetailFragment помещается в стек возврата

Класс MainActivity

Метод для отображения контакта

```
// Отображение фрагмента для добавления или изменения контакта
private void displayAddEditFragment(int viewID, Uri contactUri) {
    AddEditFragment addEditFragment = new AddEditFragment();
    // При изменении передается аргумент contactUri
    if (contactUri != null) {
        Bundle arguments = new Bundle();
        arguments.putParcelable(CONTACT_URI, contactUri);
        addEditFragment.setArguments(arguments);
    }
    // Использование FragmentTransaction для отображения
AddEditFragment
    FragmentTransaction transaction =
            getSupportFragmentManager().beginTransaction();
    transaction.replace(viewID, addEditFragment);
    transaction.addToBackStack(null);
    transaction.commit(); // Приводит к отображению AddEditFragment
}
```

Метод похож на displayContact(), но работает с фрагментом добавления/редактирования контакта AddEditFragment

Класс MainActivity

Методы DetailFragment.DetailFragmentListener

```
// Возвращение к списку контактов при удалении текущего контакта
@Override
public void onContactDeleted() {
    // Удаление с вершины стека
    getSupportFragmentManager().popBackStack();
    contactsFragment.updateContactList(); // Обновление контактов
}
```

- вызывается DetailFragment для оповещения MainActivity об удалении контакта пользователем
- DetailFragment, в котором отображалась информация о контакте, удаляется из стека
- после удаления контакта список необходимо обновить

Класс MainActivity

Методы DetailFragment.DetailFragmentListener

```
// Отображение AddEditFragment для изменения существующего контакта
@Override
public void onEditContact(Uri contactUri) {
    if (findViewById(R.id.fragmentContainer) != null) // Телефон
        displayAddEditFragment(R.id.fragmentContainer, contactUri);
    else // Планшет
        displayAddEditFragment(R.id.rightPaneContainer, contactUri);
}
```

- вызывается DetailFragment для оповещения MainActivity о редактировании контакта пользователем
- DetailFragment передает объект Uri, представляющий изменяемый контакт, чтобы его данные можно было отобразить в полях EditText фрагмента AddEditFragment для редактирования.
- для телефона AddEditFragment отображается в элементе fragmentContainer
- для планшета AddEditFragment отображается в rightPaneContainer

Класс MainActivity

Метод AddEditFragment.AddEditFragmentListener

```
    // Обновление GUI после сохранения нового или существующего контакта
    @Override
    public void onAddEditCompleted(Uri contactUri) {
        // Удаление вершины стека возврата
        getSupportFragmentManager().popBackStack();
        contactsFragment.updateContactList(); // Обновление контактов
        if (findViewById(R.id.fragmentContainer) == null) { // Планшет
            // Удаление с вершины стека возврата
            getSupportFragmentManager().popBackStack();
            // На планшете выводится добавленный или измененный контакт
            displayContact(contactUri, R.id.rightPaneContainer);
        }
    }
}
```

- вызывается AddEditFragment для оповещения MainActivity о том, что пользователь сохраняет новый контакт или сохраняет изменения в существующем контакте
- фрагмент удаляется из стека, а список контактов обновляется
- на планшете удаляется ещё один фрагмент (DetailFragment, если есть) и отображается информация о

Класс ContactsFragment

Выводит список контактов в `RecyclerView`, а также предоставляет плавающую кнопку `FloatingActionButton` для добавления нового контакта.

```
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.v4.app.Fragment;
import android.support.v4.app.LoaderManager;
import android.support.v4.content.CursorLoader;
import android.support.v4.content.Loader;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.example.someone.l5addrbook.data.DatabaseDescription.Contact;

public class ContactsFragment extends Fragment
        implements LoaderManager.LoaderCallbacks<Cursor> {
```

Класс ContactsFragment

Вложенный интерфейс

```
// Метод обратного вызова, реализуемый MainActivity
public interface ContactsFragment {
    // Вызывается при выборе контакта
    void onContactSelected(Uri contactUri);

    // Вызывается при нажатии кнопки добавления
    void onAddContact();
}
```

- методы обратного вызова реализуются MainActivity для оповещения о выборе пользователем контакта и о том, что пользователь коснулся кнопки FloatingActionButton для добавления нового контакта

Класс ContactsFragment

Поля

```
// Идентификатор Loader
private static final int CONTACTS_LOADER = 0;

// Сообщает MainActivity о выборе контакта
private ContactsFragment listener;

// Адаптер для recyclerView
private ContactsAdapter contactsAdapter;
```

- константа CONTACTS_LOADER используется для идентификации объекта Loader при обработке результатов, возвращаемых AddressBookContentProvider. В проекте используется только один объект Loader. Если их несколько, то с каждым должно быть связано уникальное целое значение для идентификации объекта в методах обратного вызова LoaderManager.LoaderCallbacks<Cursor>.
- listener ссылается на объект, реализующий интерфейс (MainActivity).
- contactsAdapter ссылается на объект ContactsAdapter,

Класс ContactsFragment

Переопределение onCreateView().

Метод заполняет и настраивает графический интерфейс

```
// Настройка графического интерфейса фрагмента
@Override
public View onCreateView(
        LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    super.onCreateView(inflater, container, savedInstanceState);
    setHasOptionsMenu(true); // У фрагмента есть команды меню

    // Заполнение GUI и получение ссылки на RecyclerView
    View view = inflater.inflate(
            R.layout.fragment_contacts, container, false);
    RecyclerView recyclerView =
            (RecyclerView) view.findViewById(R.id.recyclerView);
```

Класс ContactsFragment

Переопределение onCreateView(). Настройка RecyclerView.

```
// recyclerView выводит элементы в вертикальном списке
recyclerView.setLayoutManager(
        new LinearLayoutManager(getActivity().getBaseContext()));

// создание адаптера recyclerView и слушателя щелчков на элементах
contactsAdapter = new ContactsAdapter(
        new ContactsAdapter.ContactClickListener() {
            @Override
            public void onClick(Uri contactUri) {
                listener.onContactSelected(contactUri);
            }
        }
);
recyclerView.setAdapter(contactsAdapter); // Назначение адаптера

// Присоединение ItemDecorator для вывода разделителей
recyclerView.addItemDecoration(new ItemDivider(getContext()));

// Улучшает быстродействие,
// если размер макета RecyclerView не изменяется
recyclerView.setHasFixedSize(true);
```

Класс ContactsFragment

Переопределение onCreateView(). Настройка кнопки.

```
    // Получение FloatingActionButton и настройка слушателя
    FloatingActionButton addButton =
            (FloatingActionButton) view.findViewById(R.idaddButton);
    addButton.setOnClickListener(
            new View.OnClickListener() {
                // Отображение AddEditFragment при касании FAB
                @Override
                public void onClick(View view) {
                    listener.onAddContact();
                }
            }
    );

    return view;
}
```

Класс ContactsFragment

Переопределение onAttach() и onDetach()

```
// Присваивание ContactsFragment при присоединении фрагмента
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    listener = (ContactsFragment) context;
}

// Удаление ContactsFragment при отсоединении фрагмента
@Override
public void onDetach() {
    super.onDetach();
    listener = null;
}
```

- в качестве слушателя событий фрагмента выступает главная активность

Класс ContactsFragment

Переопределение onActivityCreated()

```
// Инициализация Loader при создании активности этого фрагмента
@Override
public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    getLoaderManager().initLoader(CONTACTS_LOADER, null, this);
}
```

- вызывается после создания управляющей активности фрагмента и завершения выполнения метода onCreateView() фрагмента
- приказывает LoaderManager инициализировать Loader (RecyclerView должен уже существовать для отображения загруженных данных)
- аргументы initLoader() :
    - целочисленный идентификатор Loader;
    - объект Bundle с аргументами конструктора Loader или null при отсутствии аргументов;
    - ссылка на реализацию интерфейса LoaderManager.LoaderCallbacks<Cursor> (представляет

Класс ContactsFragment

Метод обновления списка контактов updateContactList()

```
// Вызывается из MainActivity
// при обновлении базы данных другим фрагментом
public void updateContactList() {
    contactsAdapter.notifyDataSetChanged();
}
```

- оповещает ContactsAdapter об изменении данных.
- вызывается при добавлении новых контактов, а также обновлении или удалении существующих контактов.

Класс ContactsFragment

Методы LoaderManager.LoaderCallbacks<Cursor>

```
// Вызывается LoaderManager для создания Loader
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    // Создание CursorLoader на основании аргумента id; в этом
    // фрагменте только один объект Loader, и команда switch не нужна
    switch (id) {
        case CONTACTS_LOADER:
            return new CursorLoader(getActivity(),
                Contact.CONTENT_URI, // Uri таблицы contacts
                null, // все столбцы
                null, // все записи
                null, // без аргументов
                Contact.COLUMN_NAME + " COLLATE NOCASE ASC"); // сортировка
        default:
            return null;
    }
}
```

- LoaderManager управляет созданным объектом Loader в контексте жизненного цикла фрагмента или активности

Класс ContactsFragment

Методы LoaderManager.LoaderCallbacks<Cursor>

```
// Вызывается LoaderManager при завершении загрузки
@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    contactsAdapter.swapCursor(data);
}
```

- onLoadFinished() вызывается LoaderManager после того, как объект Loader завершит загрузку своих данных и станет возможным перейти к обработке результатов в аргументе Cursor
- метод swapCursor класса ContactsAdapter получает на вход объект Cursor, так что ContactsAdapter может обновить компонент RecyclerView на основании нового содержимого Cursor

Класс ContactsFragment

Методы LoaderManager.LoaderCallbacks<Cursor>

```
// Вызывается LoaderManager при сбросе Loader
@Override
public void onLoaderReset(Loader<Cursor> loader) {
    contactsAdapter.swapCursor(null);
}
```

- onLoaderReset() вызывается LoaderManager тогда, когда происходит сброс объекта Loader, а его данные становятся недоступными
- в этот момент приложение должно немедленно разорвать связь с данными
- метод swapCursor класса ContactsAdapter вызывается с аргументом null, показывая тем самым, что данные для связывания с RecyclerView отсутствуют

Класс ContactsAdapter

Субкласс RecyclerView.Adapter, используемый компонентом RecyclerView фрагмента ContactsFragment для связывания отсортированного списка имен контактов с RecyclerView

Дополнительные библиотеки и интерфейс

```
import android.database.Cursor;
import android.net.Uri;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import com.example.someone.l5addrbook.data.DatabaseDescription.Contact;

public class ContactsAdapter extends
    RecyclerView.Adapter<ContactsAdapter.ViewHolder> {
    // Интерфейс реализуется ContactsFragment для обработки
    // прикосновения к элементу в списке RecyclerView
    public interface ContactClickListener {
        void onClick(Uri contactUri);
    }
```

Класс ContactsAdapter

Вложенный класс ViewHolder

```
// Вложенный субкласс RecyclerView.ViewHolder используется
// для реализации паттерна View-Holder в контексте RecyclerView
public class ViewHolder extends RecyclerView.ViewHolder {
    public final TextView textView;
    private long rowID;

    // Настройка объекта ViewHolder элемента RecyclerView
    public ViewHolder(View itemView) {
        super(itemView);
        textView = (TextView) itemView.findViewById(android.R.id. text1);
        // Присоединение слушателя к itemView
        itemView.setOnClickListener(
                new View.OnClickListener() {
                    // Выполняется при щелчке на контакте в ViewHolder
                    @Override
                    public void onClick(View view) {

clickListener.onClick(Contact.buildContactUri(rowID));
                    }
                }
        );
    }
    // Идентификатор записи базы данных для контакта в ViewHolder
    public void setRowID(long rowID) {
        this.rowID = rowID;
    }
}
```

Класс ContactsAdapter

Поля и конструктор

```
// Переменные экземпляров ContactsAdapter
private Cursor cursor = null;
private final ContactClickListener clickListener;

// Конструктор
public ContactsAdapter(ContactClickListener clickListener) {
    this.clickListener = clickListener;
}
```

Класс ContactsAdapter

Переопределение onCreateViewHolder()

```
// Подготовка нового элемента списка и его объекта ViewHolder
@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    // Заполнение макета android.R.layout.simple_list_item_1
    View view = LayoutInflater.from(parent.getContext()).inflate(
            android.R.layout.simple_list_item_1, parent, false);
    return new ViewHolder(view); // ViewHolder текущего элемента
}
```

- метод заполняет графический интерфейс объекта **ViewHolder**
- используется предопределенный макет **android.R.layout.simple_list_item_1**, который определяет макет
  с одним компонентом **TextView** с именем **text1**

Класс ContactsAdapter

Переопределение onBindViewHolder()

```
// Назначает текст элемента списка
@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    cursor.moveToPosition(position);
    holder.setRowID(cursor.getLong(cursor.getColumnIndex(Contact._ID)));
    holder.textView.setText(cursor.getString(cursor.getColumnIndex(
            Contact.COLUMN_NAME)));
}
```

- метод **moveToPosition()** класса **Cursor** используется для перехода к контакту, соответствующему позиции текущего элемента **RecyclerView**
- **setRowID()** задает значение rowID для ViewHolder; метод **getColumnIndex()** класса Cursor возвращает номер поля Contact._ID, полученное число передается методу **getLong()** для получения идентификатора записи контакта
- аналогично назначается текст компонента **textView** объекта **ViewHolder** по полю Contact.COLUMN_NAME

Класс ContactsAdapter

Вспомогательные методы

```
    // Возвращает количество элементов, предоставляемых адаптером
    @Override
    public int getItemCount() {
        return (cursor != null) ? cursor.getCount() : 0;
    }

    // Текущий объект Cursor адаптера заменяется новым
    public void swapCursor(Cursor cursor) {
        this.cursor = cursor;
        notifyDataSetChanged();
    }
}
```

- **getItemCount()** возвращает общее количество строк в **Cursor** или 0, если курсор не инициализирован
- **swapCursor()** заменяет текущий объект **Cursor** адаптера и уведомляет адаптер о том, что его данные изменились. Этот метод вызывается из методов **onLoadFinished()** и **onLoaderReset()** класса **ContactsFragment**

Класс AddEditFragment

Предоставляет интерфейс для добавления новых или редактирования существующих контактов.

Дополнительные библиотеки

```
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.support.design.widget.CoordinatorLayout;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.design.widget.TextInputLayout;
import android.support.v4.app.Fragment;
import android.support.v4.app.LoaderManager;
import android.support.v4.content.CursorLoader;
import android.support.v4.content.Loader;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.inputmethod.InputMethodManager;

import com.example.someone.l5addrbook.data.DatabaseDescription.Contact;
```

Класс AddEditFragment

Суперкласс и интерфейс

```
public class AddEditFragment extends Fragment
        implements LoaderManager.LoaderCallbacks<Cursor> {

    // Определяет метод обратного вызова, реализованный MainActivity
    public interface AddEditFragmentListener {
        // Вызывается при сохранении контакта
        void onAddEditCompleted(Uri contactUri);
    }
```

- класс реализует интерфейс **LoaderManager.LoaderCallbacks<Cursor>** для реакции на события **LoaderManager**
- вложенный интерфейс **AddEditFragmentListener** содержит метод обратного вызова **onAddEditCompleted()**, реализуемый **MainActivity** для оповещения о сохранении пользователем нового или измененного существующего контакта.

Класс AddEditFragment

Поля класса

```
// Константа для идентификации Loader
private static final int CONTACT_LOADER = 0;

private AddEditFragmentListener listener; // MainActivity
private Uri contactUri; // Uri выбранного контакта
private boolean addingNewContact = true; // Добавление или изменение
```

- константа **CONTACT_LOADER** идентифицирует объект **Loader**, который обращается с запросом к **AddressBookContentProvider** для получения одного контакта для редактирования
- переменная экземпляра **listener** содержит ссылку на объект **AddEditFragmentListener** (**MainActivity**), который должен оповещаться о сохранении нового или обновленного контакта
- переменная экземпляра **contactUri** представляет редактируемый контакт
- переменная экземпляра **addingNewContact** определяет тип операции: добавление нового контакта (true) или

Класс AddEditFragment

Поля класса

```
// Компоненты EditText для информации контакта
private TextInputLayout nameTextInputLayout;
private TextInputLayout phoneTextInputLayout;
private TextInputLayout emailTextInputLayout;
private TextInputLayout streetTextInputLayout;
private TextInputLayout cityTextInputLayout;
private TextInputLayout stateTextInputLayout;
private TextInputLayout zipTextInputLayout;
private FloatingActionButton saveContactFAB;

private CoordinatorLayout coordinatorLayout; // Для SnackBar
```

- поля для доступа к интерактивным элементам фрагмента

Класс AddEditFragment

Методы жизненного цикла onAttach(), onDetach()

```
// Назначение AddEditFragmentListener при присоединении фрагмента
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    listener = (AddEditFragmentListener) context;
}
// Удаление AddEditFragmentListener при отсоединении фрагмента
@Override
public void onDetach() {
    super.onDetach();
    listener = null;
}
```

- методы **onAttach()** и **onDetach()** присваивают переменной экземпляра **listener** ссылку на управляющую активность при присоединении **AddEditFragment** или null при отсоединении **AddEditFragment**

Класс AddEditFragment

Метод onCreateView()

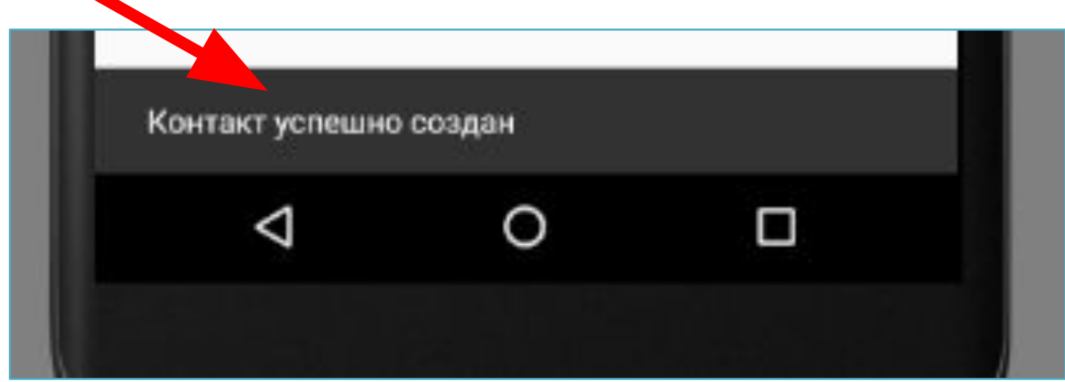
```
// Вызывается при создании представлений фрагмента
@Override
public View onCreateView(
        LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    super.onCreateView(inflater, container, savedInstanceState);
    setHasOptionsMenu( true); // У фрагмента есть команды меню

    // Заполнение GUI и получение ссылок на компоненты EditText
    View view =
            inflater.inflate(R.layout. fragment_add_edit, container, false);
    nameTextInputLayout =
            (TextInputLayout) view.findViewById(R.id. nameTextInputLayout);
    nameTextInputLayout.getEditText().addTextChangedListener(
            nameChangedListener);
    phoneTextInputLayout =
            (TextInputLayout) view.findViewById(R.id. phoneTextInputLayout);
    emailTextInputLayout =
            (TextInputLayout) view.findViewById(R.id. emailTextInputLayout);
    streetTextInputLayout =
            (TextInputLayout)
view.findViewById(R.id. streetTextInputLayout);
    cityTextInputLayout =
            (TextInputLayout) view.findViewById(R.id. cityTextInputLayout);
    stateTextInputLayout =
            (TextInputLayout) view.findViewById(R.id. stateTextInputLayout);
    zipTextInputLayout =
```

Класс AddEditFragment

Метод onCreateView()

```
// Назначение слушателя событий FloatingActionButton
saveContactFAB = (FloatingActionButton) view.findViewById(
        R.id.saveFloatingActionButton);
saveContactFAB.setOnClickListener(saveContactButtonClicked);
updateSaveButtonFAB();

// Используется для отображения SnackBar с короткими сообщениями
coordinatorLayout = (CoordinatorLayout) getActivity().findViewById(
        R.id.coordinatorLayout);
```

- **FloatingActionButton** – кнопка сохранения
- **SnackBar** используется для информирования о результатах действий пользователя



Класс AddEditFragment

Метод onCreateView()

```
    Bundle arguments = getArguments(); // null при создании контакта
    if (arguments != null) {
        addingNewContact = false;
        contactUri = arguments.getParcelable(MainActivity.CONTACT_URI);
    }
    // При изменении существующего контакта создать Loader
    if (contactUri != null)
        getLoaderManager().initLoader(CONTACT_LOADER, null, this);
    return view;
}
```

- при добавлении нового контакта вместо **Bundle** в **onCreateView()** передаётся null
- в противном случае получаем URI редактируемого контакта **contactUri**
- если значение **contactUri** отлично от null, используем объект **LoaderManager** фрагмента для инициализации объекта **Loader**, который будет использоваться **AddEditFragment** для получения данных редактируемого контакта

Класс AddEditFragment

Отслеживание изменений в полях формы

```
private final TextWatcher nameChangedListener = new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count,
                                  int after) {
    }
    // Вызывается при изменении текста в nameTextInputLayout
    @Override
    public void onTextChanged(CharSequence s, int start, int before,
                              int count) {
        updateSaveButtonFAB();
    }
    @Override
    public void afterTextChanged(Editable s) {
    }
};
```

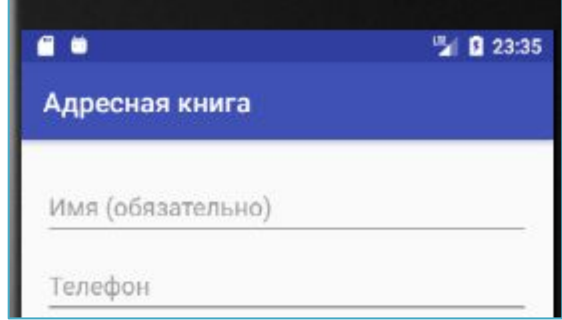
- метод **onTextChanged()** вызывается при любом изменении текстовых полей с элементами описания контакта

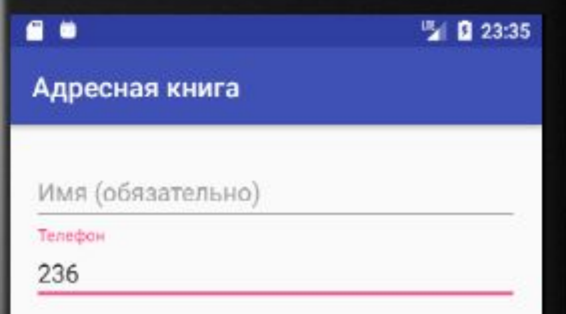
Класс AddEditFragment

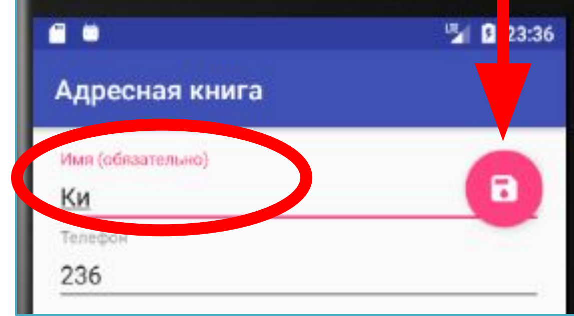
Отслеживание изменений в полях формы

```
// Кнопка saveButtonFAB видна, если имя не пусто
private void updateSaveButtonFAB() {
    String input =
            nameTextInputLayout.getEditText().getText().toString();

    // Если для контакта указано имя, показать FloatingActionButton
    if (input.trim().length() != 0)
        saveContactFAB.show();
    else
        saveContactFAB.hide();
}
```

- в описании контакта обязательным является только имя, поэтому кнопка сохранения **saveContactFAB** отображается только при ненулевой длине имени контакта







Класс AddEditFragment

Слушатель для кнопки сохранения

```
private final View.OnClickListener saveContactButtonClicked =
    new View.OnClickListener()
    {
        @Override
        public void onClick(View v) {
            // Скрыть виртуальную клавиатуру
            ((InputMethodManager) getActivity().getSystemService(
                Context.INPUT_METHOD_SERVICE)).hideSoftInputFromWindow(
                getView().getWindowToken(), 0);
            saveContact(); // Сохранение контакта в базе данных
        }
    };
```

- метод **onClick()** скрывает виртуальную клавиатуру, а затем вызывает метод **saveContact()** для сохранения контакта

Класс AddEditFragment

Сохранение информации о контакте

```
private void saveContact() {
    // Создание объекта ContentValues с парами "ключ—значение"
    ContentValues contentValues = new ContentValues();
    contentValues.put(Contact.COLUMN_NAME,
            nameTextInputLayout.getEditText().getText().toString());
    contentValues.put(Contact.COLUMN_PHONE,
            phoneTextInputLayout.getEditText().getText().toString());
    contentValues.put(Contact.COLUMN_EMAIL,
            emailTextInputLayout.getEditText().getText().toString());
    contentValues.put(Contact.COLUMN_STREET,
            streetTextInputLayout.getEditText().getText().toString());
    contentValues.put(Contact.COLUMN_CITY,
            cityTextInputLayout.getEditText().getText().toString());
    contentValues.put(Contact.COLUMN_STATE,
            stateTextInputLayout.getEditText().getText().toString());
    contentValues.put(Contact.COLUMN_ZIP,
            zipTextInputLayout.getEditText().getText().toString());
```

- **contentValues** содержит имена столбцов и значения, которые должны вставляться или обновляться в базе данных; имена полей получаем из свойств статического объекта **Contact**

Класс AddEditFragment

Сохранение информации о контакте (новый контакт)

```
if (addingNewContact) {
    // Использовать объект ContentResolver активности для вызова
    // insert для объекта AddressBookContentProvider
    Uri newContactUri = getActivity().getContentResolver().insert(
            Contact.CONTENT_URI, contentValues);

    if (newContactUri != null) {
        Snackbar.make(coordinatorLayout,
            R.string.contact_added, Snackbar.LENGTH_LONG).show();
        listener.onAddEditCompleted(newContactUri);

    } else {
        Snackbar.make(coordinatorLayout,
            R.string.contact_not_added,Snackbar.LENGTH_LONG).show();
    }
}
```

- для добавления контакта в БД вызывается метод **insert()** провайдера **AddressBookContentProvider**
- об успешном добавлении оповещается слушатель (**MainActivity**)
- результат добавления (успех/ошибка) сообщается пользователю через объект **Snackbar**

Класс AddEditFragment

Сохранение информации о контакте (редактирование

```
  else {
     // Использовать объект ContentResolver активности для вызова
     // update для объекта AddressBookContentProvider
     int updatedRows = getActivity().getContentResolver().update(
          contactUri, contentValues, null, null);
     if (updatedRows > 0) {
          listener.onAddEditCompleted(contactUri);
          Snackbar.make(coordinatorLayout,
               R.string.contact_updated, Snackbar.LENGTH_LONG).show();

     } else {
          Snackbar.make(coordinatorLayout,
               R.string.contact_not_updated, Snackbar.LENGTH_LONG).show();
     }
  }
}
```

- для изменения контакта в БД вызывается метод **update()** провайдера **AddressBookContentProvider**
- об успешном добавлении оповещается слушатель (**MainActivity**)
- результат добавления (успех/ошибка) сообщается

Класс AddEditFragment

Методы интерфейса LoadManager (onCreateLoader())

```
// Вызывается LoaderManager для создания Loader
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    // Создание CursorLoader на основании аргумента id; в этом
    // фрагменте только один объект Loader, и команда switch не нужна
    switch (id) {
        case CONTACT_LOADER:
            return new CursorLoader(getActivity(),
                    contactUri, // Uri отображаемого контакта
                    null, // Все столбцы
                    null, // Все записи
                    null, // Без аргументов
                    null); // Порядок сортировки
        default:
            return null;
    }
}
```

- **Loader** используется только при редактировании контакта (метод onCreateView())
- **Loader** создаётся для конкретного редактируемого контакта

Класс AddEditFragment

Методы интерфейса LoadManager (onLoadFinished())

```
// Вызывается LoaderManager при завершении загрузки
@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    // Если контакт существует в базе данных, вывести его информацию
    if (data != null && data.moveToFirst()) {
        // Получение индекса столбца для каждого элемента данных
        int nameIndex = data.getColumnIndex(Contact.COLUMN_NAME);
        int phoneIndex = data.getColumnIndex(Contact.COLUMN_PHONE);
        int emailIndex = data.getColumnIndex(Contact.COLUMN_EMAIL);
        int streetIndex = data.getColumnIndex(Contact.COLUMN_STREET);
        int cityIndex = data.getColumnIndex(Contact.COLUMN_CITY);
        int stateIndex = data.getColumnIndex(Contact.COLUMN_STATE);
        int zipIndex = data.getColumnIndex(Contact.COLUMN_ZIP);
```

- если контакт существует в БД, то получаем всю информацию о нём из курсора…

Класс AddEditFragment

Методы интерфейса LoadManager (onLoadFinished())

```
// Вызывается LoaderManager при завершении загрузки
// Заполнение компонентов EditText полученными данными
        nameTextInputLayout.getEditText().setText(
                data.getString(nameIndex));
        phoneTextInputLayout.getEditText().setText(
                data.getString(phoneIndex));
        emailTextInputLayout.getEditText().setText(
                data.getString(emailIndex));
        streetTextInputLayout.getEditText().setText(
                data.getString(streetIndex));
        cityTextInputLayout.getEditText().setText(
                data.getString(cityIndex));
        stateTextInputLayout.getEditText().setText(
                data.getString(stateIndex));
        zipTextInputLayout.getEditText().setText(
                data.getString(zipIndex));
        updateSaveButtonFAB();
    }
}
```

- … и отображаем информацию на экране
- обновляем состояние кнопки сохранения

Класс AddEditFragment

Методы интерфейса LoadManager (onLoaderReset())

```
    // Вызывается LoaderManager при сбросе Loader
    @Override
    public void onLoaderReset(Loader<Cursor> loader) {
    }

} // окончание класса
```

- метод **onLoaderReset()** не используется в **AddEditFragment**, но должен быть переопределён

Класс DetailFragment

Выводит информацию одного контакта и предоставляет команды меню на панели приложения, при помощи которых пользователь может изменить или удалить данные контакта.

Дополнительные библиотеки

```
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.Context;
import android.content.DialogInterface;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.support.v4.app.Fragment;
import android.support.v4.app.LoaderManager;
import android.support.v4.content.CursorLoader;
import android.support.v4.content.Loader;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import java.io.Serializable;
import
com.example.someone.l5addrbook.data.DatabaseDescription.Contact;
```

Класс DetailFragment

Суперкласс и интерфейс

```
public class DetailFragment extends Fragment
      implements LoaderManager.LoaderCallbacks<Cursor> {
   // Методы обратного вызова, реализованные MainActivity
   public interface DetailFragmentListener {
      void onContactDeleted(); // Вызывается при удалении контакта

      // Передает URI редактируемого контакта DetailFragmentListener
      void onEditContact(Uri contactUri);
   }
```

- аналогично **AddEditFragment**, класс реализует интерфейс **LoaderManager.LoaderCallbacks<Cursor>** для реакции на события **LoaderManager**
- вложенный интерфейс **DetailFragmentListener** содержит методы обратного вызова, реализуемые **MainActivity** для оповещения об удалении контакта и касании команды меню для редактирования контакта

Класс DetailFragment

Поля класса

```
private static final int CONTACT_LOADER = 0; // Идентифицирует Loader

private DetailFragmentListener listener; // MainActivity
private Uri contactUri; // Uri выбранного контакта

private TextView nameTextView; // Имя контакта
private TextView phoneTextView; // Телефон
private TextView emailTextView; // Электронная почта
private TextView streetTextView; // Улица
private TextView cityTextView; // Город
private TextView stateTextView; // Штат
private TextView zipTextView; // Почтовый индекс
```

- **CONTACT_LOADER** идентифицирует объект **Loader**, который обращается с запросом к **AddressBookContentProvider** для получения одного контакта для отображения
- **listener** содержит ссылку на объект **DetailFragmentListener** (**MainActivity**), который должен оповещаться об удалении контакта или начале его редактирования
- **contactUri** представляет отображаемый контакт
- остальные поля содержат ссылки на компоненты **TextView** фрагмента

Класс DetailFragment

Методы жизненного цикла onAttach(), onDetach()

```
// Назначение DetailFragmentListener при присоединении фрагмента
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    listener = (DetailFragmentListener) context;
}

// Удаление DetailFragmentListener при отсоединении фрагмента
@Override
public void onDetach() {
    super.onDetach();
    listener = null;
}
```

- методы **onAttach()** и **onDetach()** присваивают переменной экземпляра **listener** ссылку на управляющую активность при присоединении **DetailFragment** или null при отсоединении **DetailFragment**

Класс DetailFragment

Метод onCreateView()

```
// Вызывается при создании представлений фрагмента
@Override
public View onCreateView(
        LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
  super.onCreateView(inflater, container, savedInstanceState);
  setHasOptionsMenu(true); // У фрагмента есть команды меню

  // Получение объекта Bundle с аргументами и извлечение URI
  Bundle arguments = getArguments();

  if (arguments != null)
      contactUri = arguments.getParcelable(MainActivity.CONTACT_URI);

  // Заполнение макета DetailFragment
  View view =
      inflater.inflate(R.layout.fragment_details, container, false);
```

- через объект Bundle получаем URI выбранного контакта
- заполняем макет

Класс DetailFragment

Метод onCreateView()

```
    // Получение компонентов EditTexts
    nameTextView = (TextView) view.findViewById(R.id.nameTextView);
    phoneTextView = (TextView) view.findViewById(R.id.phoneTextView);
    emailTextView = (TextView) view.findViewById(R.id.emailTextView);
    streetTextView = (TextView) view.findViewById(R.id.streetTextView);
    cityTextView = (TextView) view.findViewById(R.id.cityTextView);
    stateTextView = (TextView) view.findViewById(R.id.stateTextView);
    zipTextView = (TextView) view.findViewById(R.id.zipTextView);

    // Загрузка контакта
    getLoaderManager().initLoader(CONTACT_LOADER, null, this);
    return view;
}
```

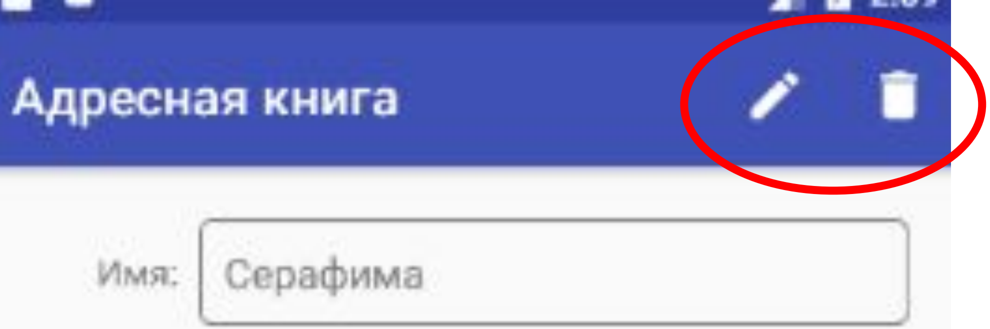
- получаем ссылки на компоненты **TextView**
- объект **LoaderManager** фрагмента используется для инициализации объекта **Loader**, который будет получать данные отображаемого контакта.

Класс DetailFragment

Отображение команд меню фрагмента

```
// Отображение команд меню фрагмента
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_details_menu, menu);
}
```

- меню заполняем по ресурсному файлу **fragment_details_menu**



Класс DetailFragment

Обработка выбора команд меню

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_edit:
            listener.onEditContact(contactUri); // Передача Uri слушателю
            return true;
        case R.id.action_delete:
            deleteContact();
            return true;
    }
    return super.onOptionsItemSelected(item);
}
```

- при выборе опции «Редактировать» URI контакта и управление передаются методу обратного вызова **onEditContact()**
- при удалении управление передаётся методу **deleteContact()**

Класс DetailFragment

Удаление контакта

```
public static class ConfirmDialogFragment extends DialogFragment {
    public static ConfirmDialogFragment newInstance(Uri uri,
                                     Context context) {
        ConfirmDialogFragment frag = new ConfirmDialogFragment();
        Bundle args = new Bundle();
        args.putParcelable(MainActivity.CONTACT_URI,uri);
        args.putSerializable("listener",(Serializable)context);
        frag.setArguments(args);
        return frag;
    }
```

- **ConfirmDialogFragment** – вспомогательный диалог для подтверждения операции удаления
- наследники **DialogFragment** должны быть статическими
- внутри статических классов Java запрещает доступ к членам вмещающего класса (здесь – к полю **listener**)
- для передачи ссылки на активность listener используется универсальный интерфейс **Serializable** (его реализует MainActivity)
- объект класса **Bundle** используется для передачи данных между методами диалога

Класс DetailFragment

Удаление контакта

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    final Uri uri=getArguments().getParcelable(MainActivity.CONTACT_URI);
    final DetailFragmentListener listener =
      (DetailFragmentListener)getArguments().getSerializable("listener");
```

- при создании диалога получаем через объект **Bundle** информацию об URI выбранного контакта и об активности-слушателе для обращения к методу обратного вызова
- для получения ссылки на активность listener как на слушателя методов обратного вызова необходимо явное приведение типа к интерфейсу **DetailFragmentListener**

Класс DetailFragment

Удаление контакта

```
    return new AlertDialog.Builder(getActivity())
            .setTitle(R.string.confirm_title)
            .setMessage(R.string.confirm_message)
            .setPositiveButton(R.string.button_delete,
                    new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(
                                DialogInterface dialog, int button) {
                            // объект ContentResolver используется
                            // для вызова delete в AddressBookContentProvider
                            getActivity().getContentResolver().delete(
                                    uri, null, null);
                            listener.onContactDeleted(); //Оповещение слушателя
                        }
                    }
            )
            .create();
    }
}
```

- при подтверждении удаления через **ContentResolver** вызывается метод **delete()** класса **AddressBookContentProvider**, удаляющий контакт из базы данных
- идентификатор записи удаляемого контакта встроен в URI
- метод **onContactDeleted()** вызывается для удаления **DetailFragment** с экрана

Класс DetailFragment

Удаление контакта

```
// DialogFragment для подтверждения удаления контакта
private DialogFragment confirmDelete;

// Удаление контакта
private void deleteContact() {
    // FragmentManager используется для отображения confirmDelete
    confirmDelete=ConfirmDialogFragment.newInstance(contactUri,
                                    (Context)listener);
    confirmDelete.show(getFragmentManager(), "confirm delete");
}
```

- метод **deleteContact()** создаёт экземпляр диалога подтверждения удаления, передавая туда URI удаляемого контакта и ссылку на главную активность, а затем открывает этот диалог

Класс DetailFragment

Удаление контакта

```
        // Кнопка OK просто закрывает диалоговое окно
        builder.setPositiveButton(R.string.button_delete,
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(
                    DialogInterface dialog, int button) {
                    // объект ContentResolver используется
                    // для вызова delete в AddressBookContentProvider
                    getActivity().getContentResolver().delete(
                            contactUri, null, null);
                    listener.onContactDeleted(); // Оповещение слушателя
                }
            }
        );
        builder.setNegativeButton(R.string.button_cancel, null);
        return builder.create(); // Вернуть AlertDialog
    }
};
```

- при подтверждении удаления через **ContentResolver** вызывается метод **delete()** класса **AddressBookContentProvider**, удаляющий контакт из базы данных
- идентификатор записи удаляемого контакта встроен в URI
- метод **onContactDeleted()** вызывается для удаления **DetailFragment** с экрана

Класс DetailFragment

Методы интерфейса LoadManager (onCreateLoader())

```
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
CursorLoader cursorLoader;
    switch (id) {
        case CONTACT_LOADER:
            cursorLoader = new CursorLoader(getActivity(),
                    contactUri, // Uri отображаемого контакта
                    null, // Все столбцы
                    null, // Все записи
                    null, // Без аргументов
                    null); // Порядок сортировки
            break;
        default:
            cursorLoader = null;
            break;
    }
    return cursorLoader;
}
```

- **Loader** создаётся и используется при загрузке конкретного контакта

Класс DetailFragment

Методы интерфейса LoadManager (onLoadFinished())

```
// Вызывается LoaderManager при завершении загрузки
@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    // Если контакт существует в базе данных, вывести его информацию
    if (data != null && data.moveToFirst()) {
        // Получение индекса столбца для каждого элемента данных
        int nameIndex = data.getColumnIndex(Contact.COLUMN_NAME);
        int phoneIndex = data.getColumnIndex(Contact.COLUMN_PHONE);
        int emailIndex = data.getColumnIndex(Contact.COLUMN_EMAIL);
        int streetIndex = data.getColumnIndex(Contact.COLUMN_STREET);
        int cityIndex = data.getColumnIndex(Contact.COLUMN_CITY);
        int stateIndex = data.getColumnIndex(Contact.COLUMN_STATE);
        int zipIndex = data.getColumnIndex(Contact.COLUMN_ZIP);
```

- если контакт существует в БД, то получаем всю информацию о нём из курсора…

Класс DetailFragment

Методы интерфейса LoadManager (onLoadFinished())

```
        // Заполнение TextView полученными данными
        nameTextView.setText(data.getString(nameIndex));
        phoneTextView.setText(data.getString(phoneIndex));
        emailTextView.setText(data.getString(emailIndex));
        streetTextView.setText(data.getString(streetIndex));
        cityTextView.setText(data.getString(cityIndex));
        stateTextView.setText(data.getString(stateIndex));
        zipTextView.setText(data.getString(zipIndex));
    }
}
```

- … и отображаем информацию на экране

Класс DetailFragment

Методы интерфейса LoadManager (onLoaderReset())

```
    // Вызывается LoaderManager при сбросе Loader
    @Override
    public void onLoaderReset(Loader<Cursor> loader) {
    }

} // окончание класса
```

- метод **onLoaderReset()** не используется в **DetailFragment**, но должен быть переопределён

Класс ItemDivider

Класс рисует разделительные линии между элементами списка в компоненте RecyclerView. Является наследником класса RecyclerView.ItemDecorator, предназначенного для создания декоративных элементов в RecyclerView.

Дополнительные библиотеки, суперкласс и конструктор

```
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.drawable.Drawable;
import android.support.v7.widget.RecyclerView;
import android.view.View;

class ItemDivider extends RecyclerView.ItemDecoration {
    private final Drawable divider;
    // Конструктор загружает встроенный разделитель элементов списка
    public ItemDivider(Context context) {
        int[] attrs = {android.R.attr.listDivider};
        divider = context.obtainStyledAttributes(attrs).getDrawable(0);
    }
```

Класс ItemDivider

Метод onDrawOver() для рисования разделителей

```
// Рисование разделителей элементов списка в RecyclerView
@Override
public void onDrawOver(Canvas c, RecyclerView parent,
                                RecyclerView.State state) {
    super.onDrawOver(c, parent, state);
```

- В процессе прокрутки **RecyclerView** содержимое списка постоянно перерисовывается в новых позициях экрана. Частью процесса перерисовки является вызов метода **onDrawOver()** объекта **RecyclerView.ItemDecoration**
- Аргументы метода:
  - Canvas - холст для рисования декоративных элементов
  - RecyclerView - объект, на котором рисуется содержимое Canvas
  - RecyclerView.State - объект с информацией, передаваемой между разными компонентами RecyclerView. В этом приложении значение просто

Класс ItemDivider

Метод onDrawOver() для рисования разделителей

```
// Вычисление координат x для всех разделителей
int left = parent.getPaddingLeft();
int right = parent.getWidth() - parent.getPaddingRight();
```

- **left** и **right** - левая и правая координаты x, определяющие границы выводимого объекта **Drawable**
- **getPaddingLeft()** возвращает величину отступа между левым краем **RecyclerView** и его содержимым
- **getWidth()** возвращает ширину компонента **RecyclerView**
- **getPaddingRight()** возвращает величину отступа между правым краем **RecyclerView** и его содержимым.

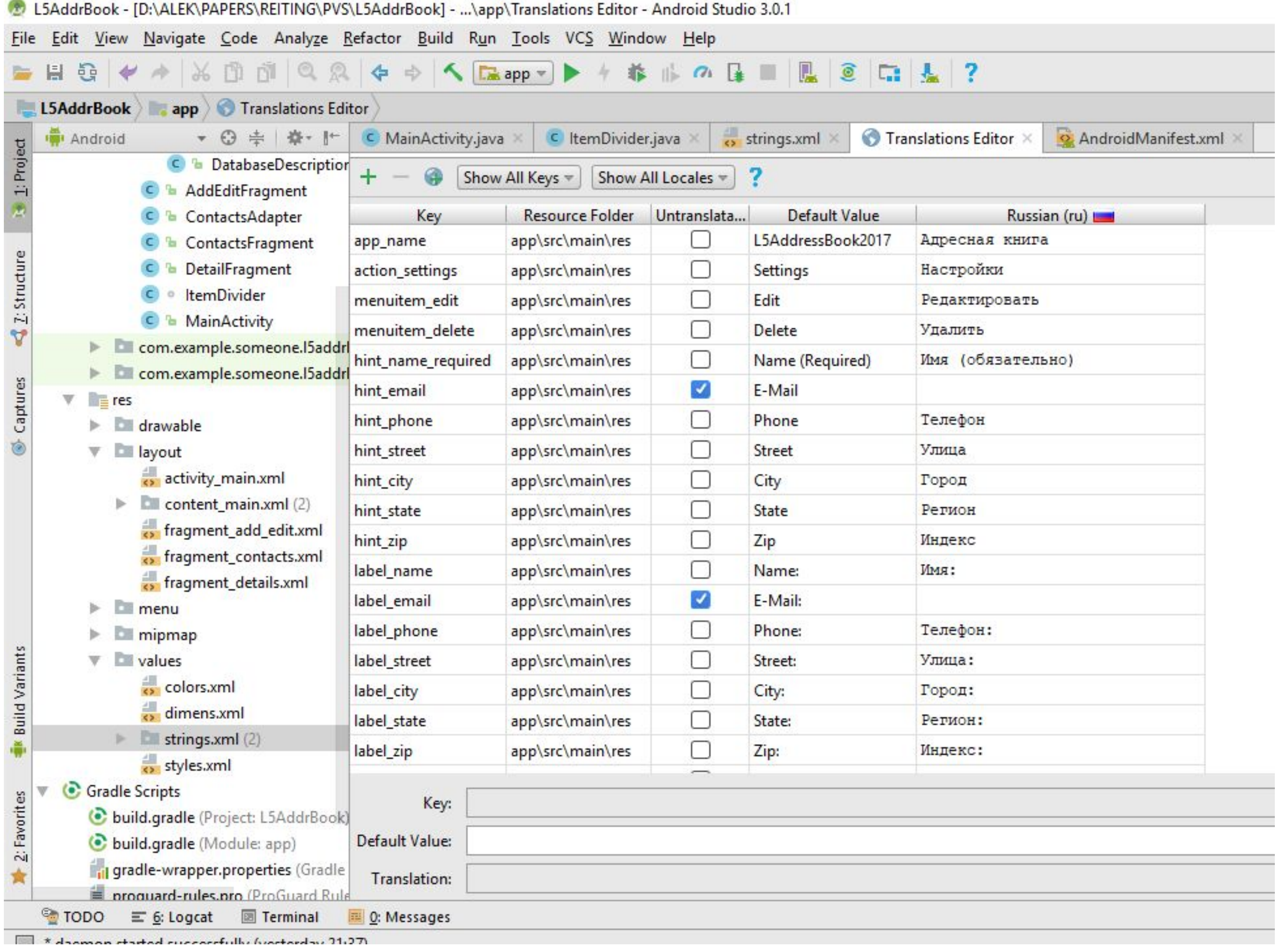
Класс ItemDivider

Метод onDrawOver() для рисования разделителей

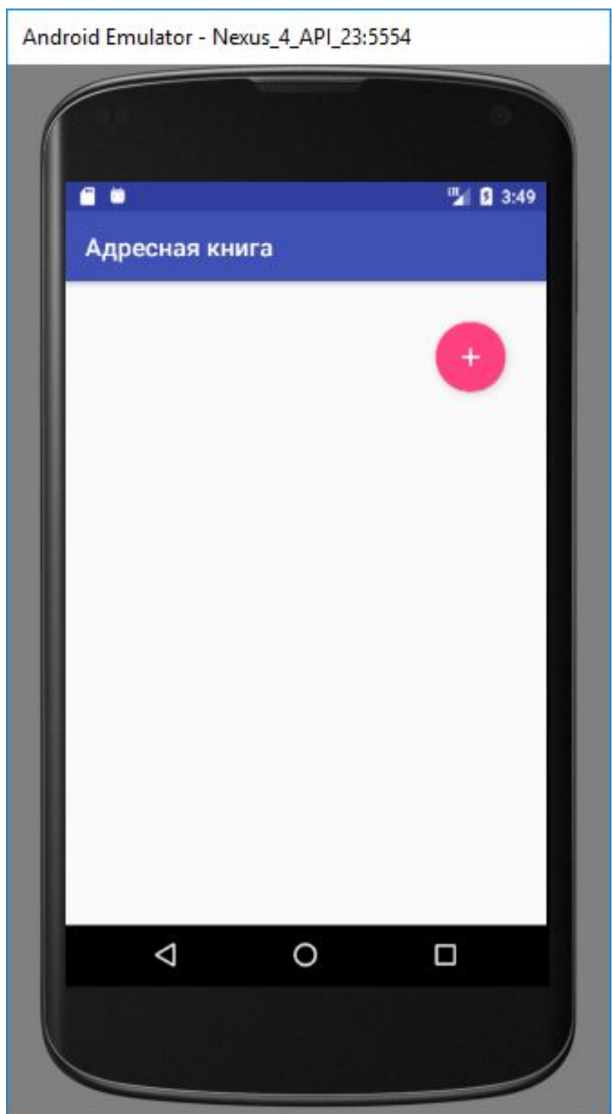
```
    // Для каждого элемента, кроме последнего, нарисовать линию
    for (int i = 0; i < parent.getChildCount() - 1; ++i) {
        View item = parent.getChildAt(i); // Получить i-й элемент списка

        // Вычисление координат y текущего разделителя
        int top = item.getBottom() + ((RecyclerView.LayoutParams)
                item.getLayoutParams()).bottomMargin;
        int bottom = top + divider.getIntrinsicHeight();

        // Рисование разделителя с вычисленными границами
        divider.setBounds(left, top, right, bottom);
        divider.draw(c);
    }
  }
}
```

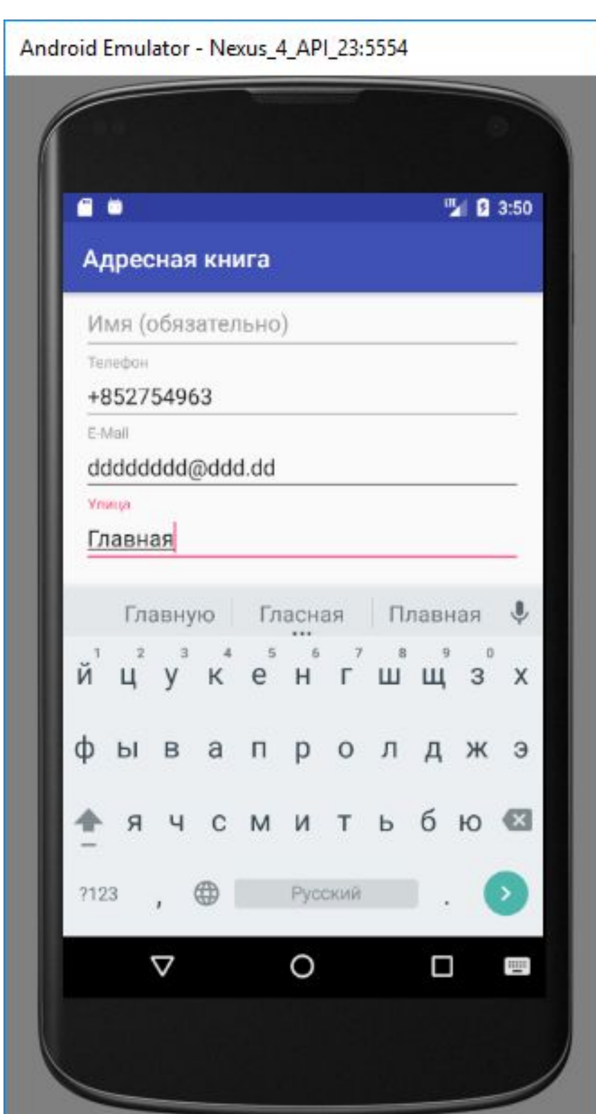
- разделитель выводится под каждым элементом, кроме последнего
- **getBottom()**, **getIntrinsicHeight()**, **bottomMargin** позволяют вычислить координаты по оси y

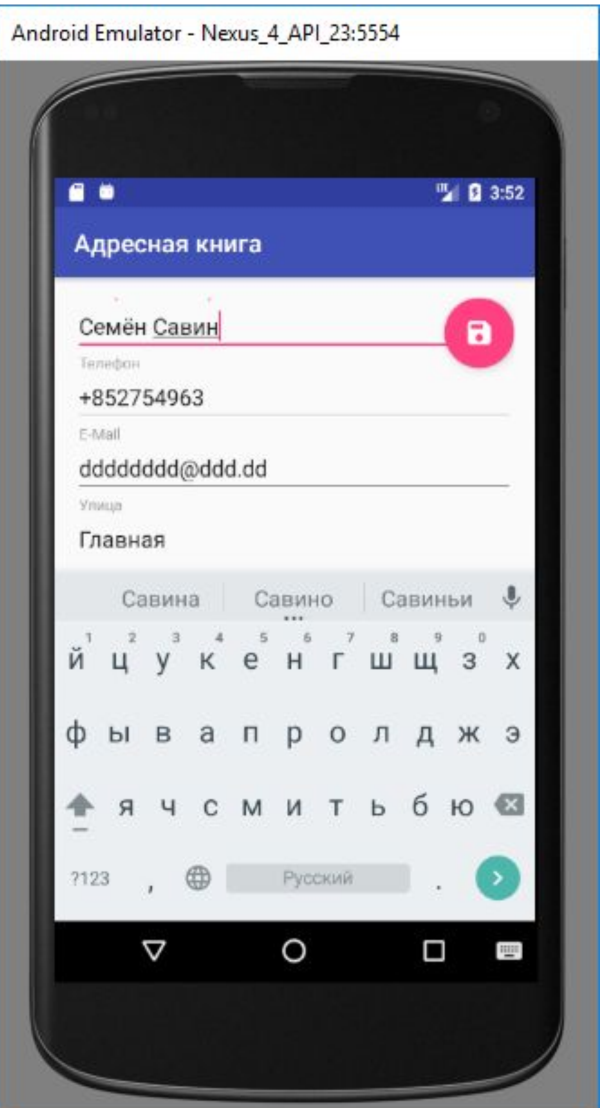
Интернационализация



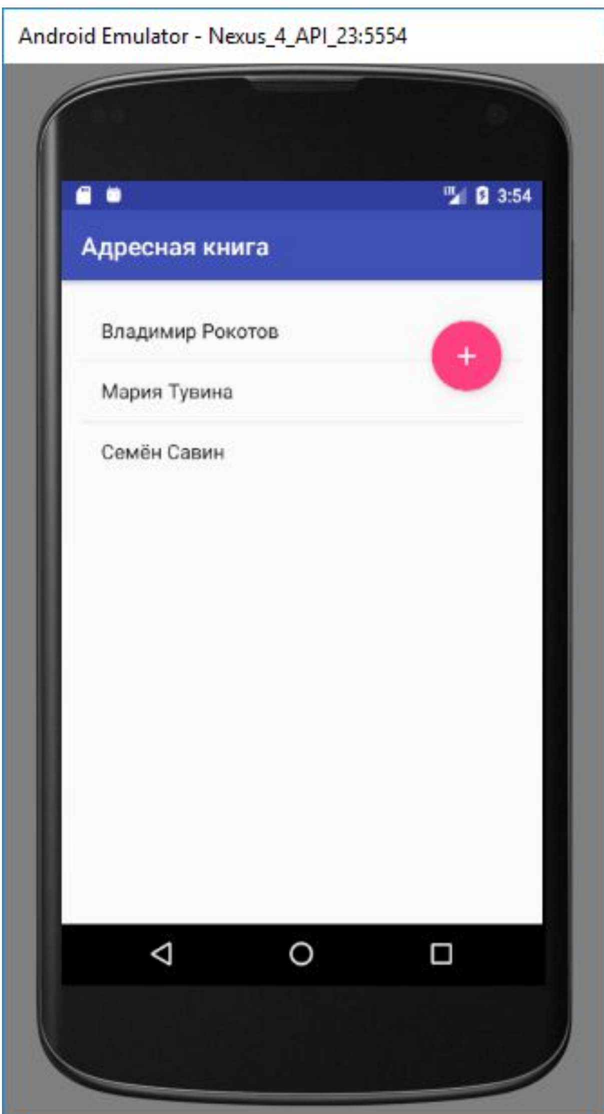
Проверка работоспособности

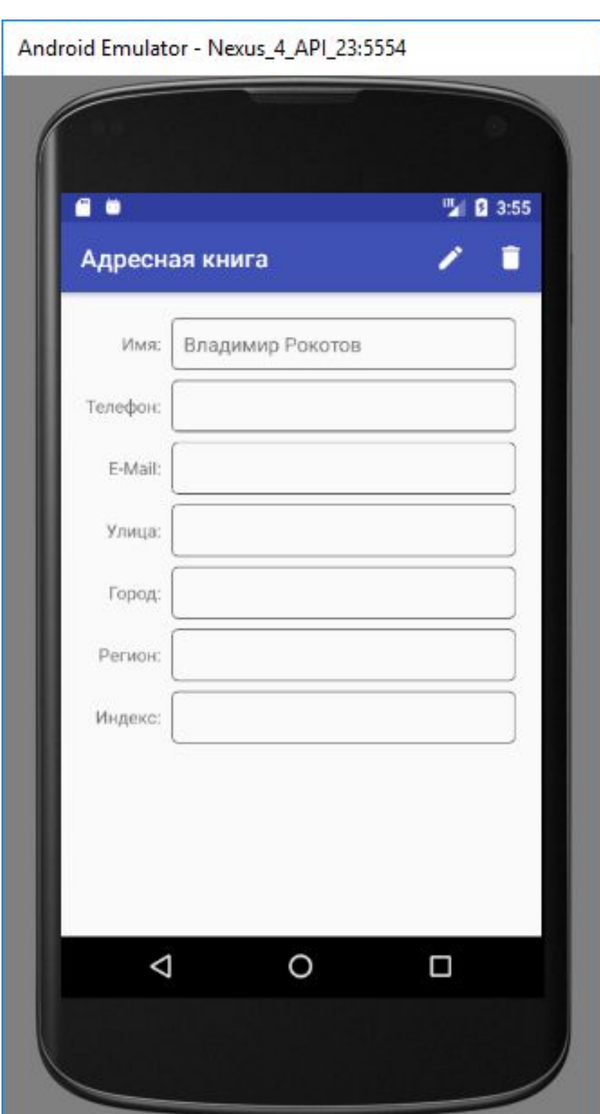


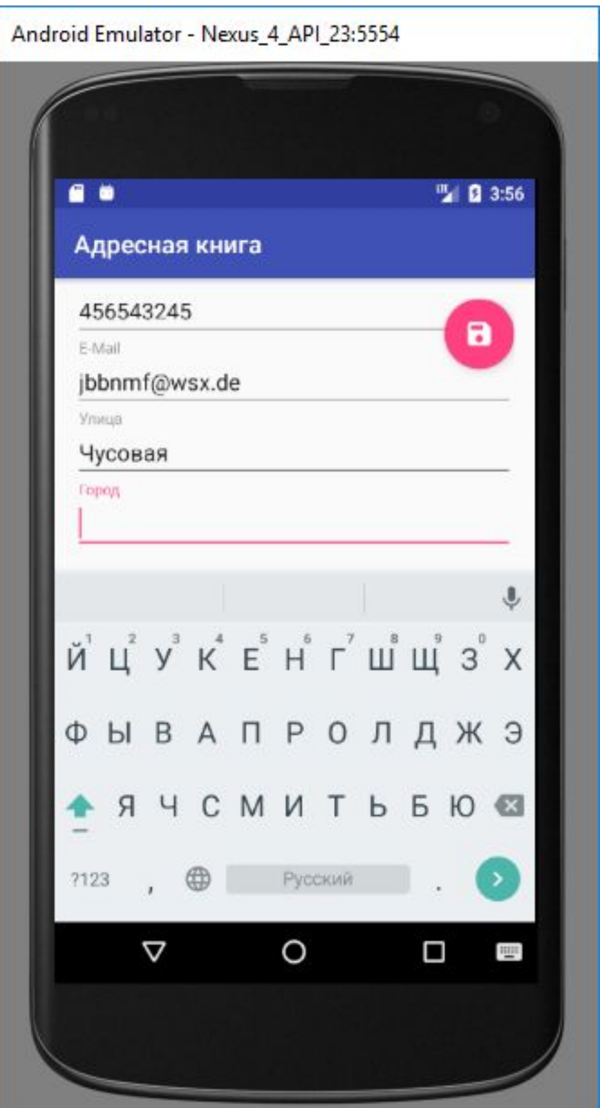




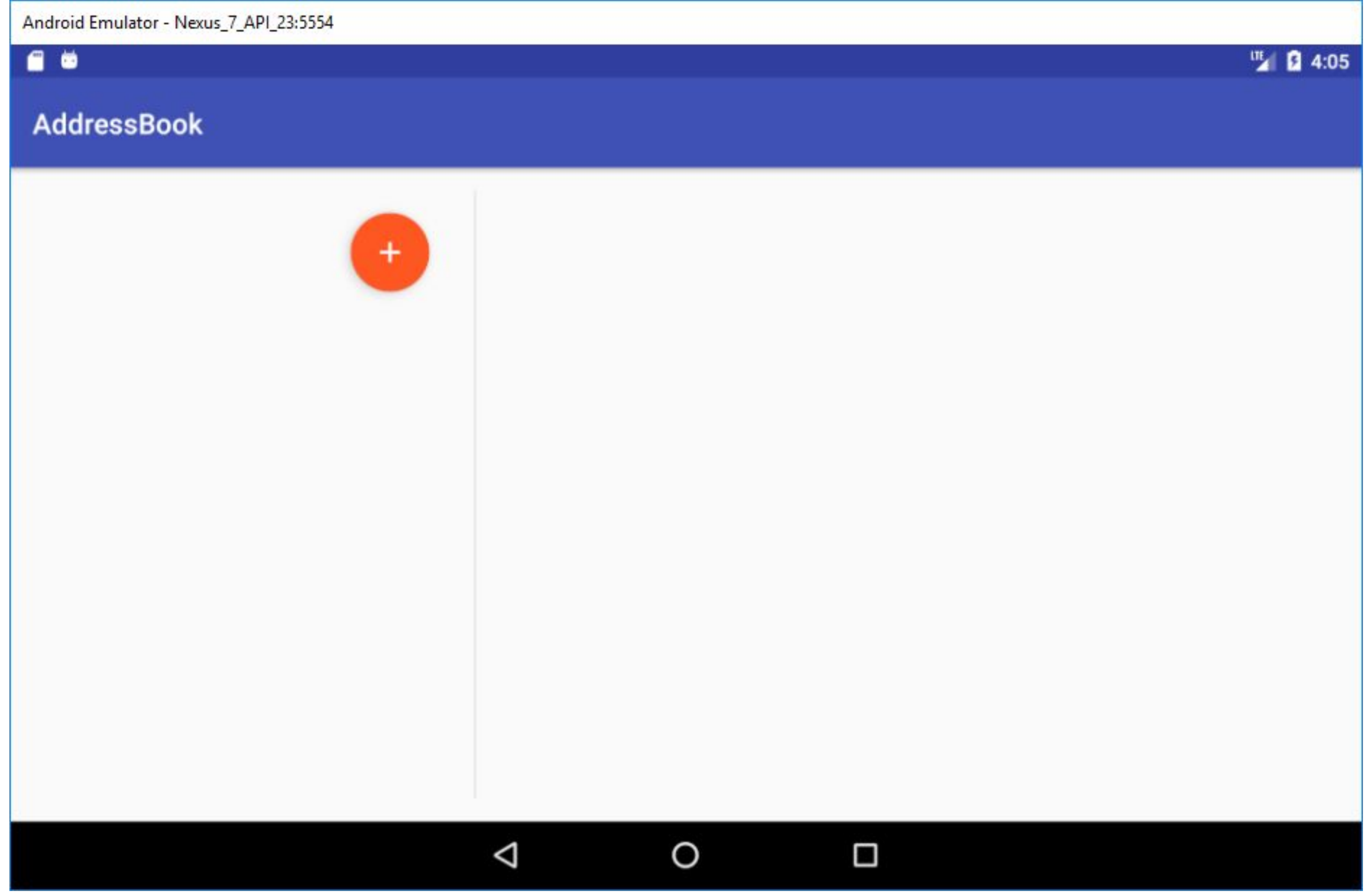
Проверка работоспособности



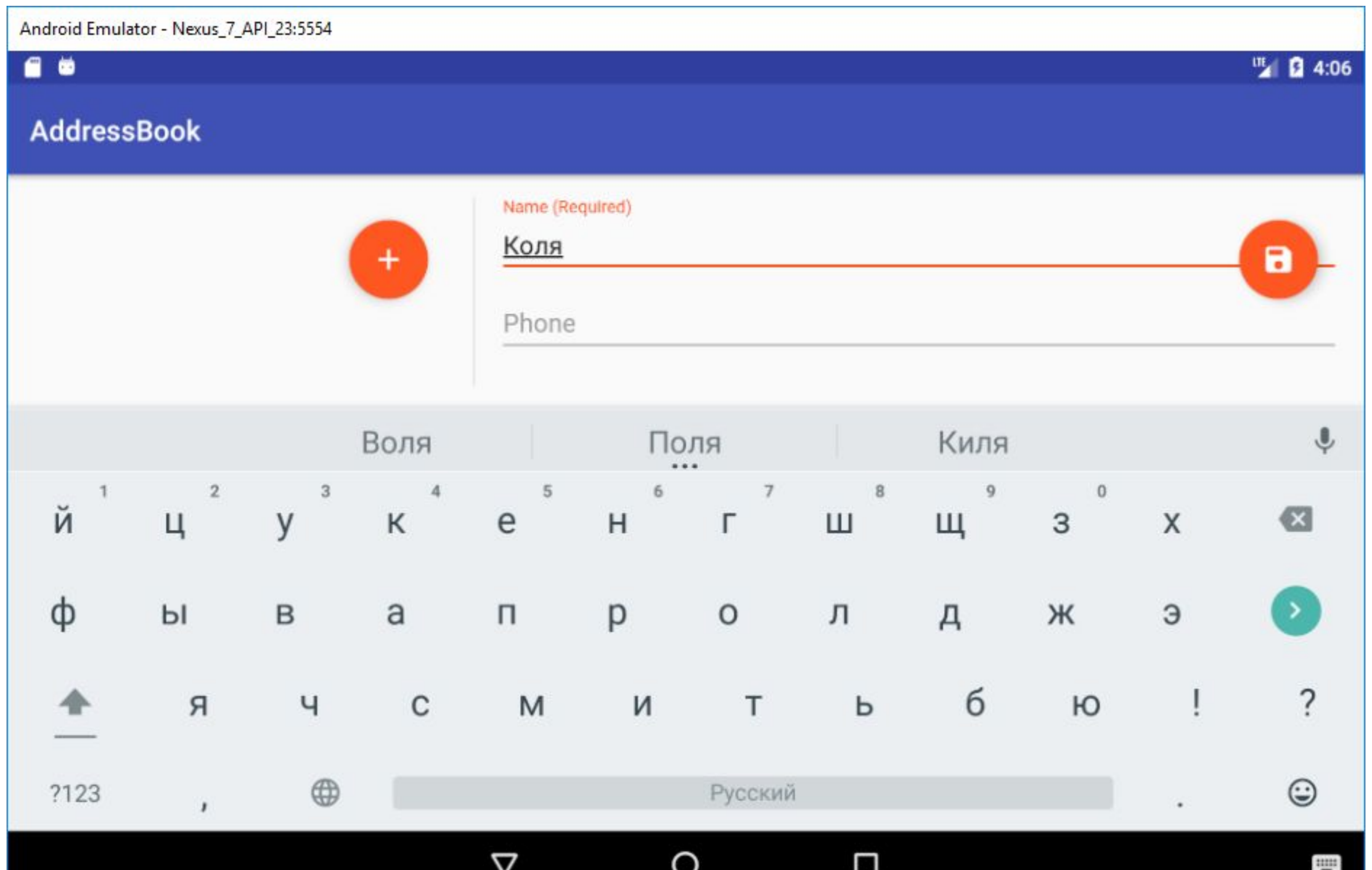




Проверка на планшете



Проверка на планшете



Проверка на планшете

Android Emulator - Nexus_7_API_23:5554

4:07

AddressBook

Коля

Оля

Толя

Name: Толя

Phone:

E-Mail:

Street:

City:

State:

Zip:

Настройки Gradle

L5AddrBook - [D:\ALEK\PAPERS\REITING\PVS\L5AddrBook] - app - Android Studio 3.0.1

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

L5AddrBook > app > build.gradle

- app
- Gradle Scripts
  - build.gradle (Project: L5AddrBook)
  - build.gradle (Module: app)
  - gradle-wrapper.properties (Gradle
  - proguard-rules.pro (ProGuard Rule
  - gradle.properties (Project Properti
  - settings.gradle (Project Settings)
  - local.properties (SDK Location)

MainActivity.java | ItemDivider.java | strings.xml | L5AddrBook | app | AndroidManifest.xml

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "com.example.someone.l5addrbook"
        minSdkVersion 23
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_7
        targetCompatibility JavaVersion.VERSION_1_7
    }
    productFlavors {
    }
    buildToolsVersion '26.0.2'
}
dependencies {
    implementation fileTree(include: ['*.jar'], dir: 'libs')
    testImplementation 'junit:junit:4.12'
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support:design:26.1.0'
    implementation 'com.android.support:recyclerview-v7:26.1.0'
    implementation 'com.android.support:support-v4:26.1.0'
}
```

Проверить соответствие версий между библиотеками!

Приложение

- описание строковых ресурсов (английский)
- описание строковых ресурсов (русский)
- класс MainActivity
- класс ContactsFragment
- класс DetailFragment
- класс AddEditFragment
- класс ContactsAdapter
- класс ItemDivider
- класс DatabaseDescription
- класс AddressBookDatabaseHelper
- класс AddressBookContentProvider