Разработка 3 простых приложений на Android

Практический мини курс

От сайта

Prologistic.com.ua

Разработка 3 простых приложений на Android

Быстрый старт в практической разработке

Начинающему Android-разработчику требуется приложить множество усилий и времени, чтобы с нуля написать свое первое приложение под Андроид (Hello World не считается ☺). Мне это очень знакомо, так как в свое время написание чего-то с нуля с пользовательским интерфейсом и базой данных часто приводило меня в ступор. Я не знал с чего начать и с какой стороны подойти к реализации. По началу, я читал книги по программированию под Android, но они требовали много свободного времени. Также я читал уроки в Интернете, но они были либо слишком сложны для меня, как начинающего, либо уже устаревшими.

Поэтому я решил сделать небольшую книжечку, в которой будет описано создание с нуля 3 простых, но рабочих приложений. Они затрагивают лишь малую часть Android Framework, но дают хороший старт и самое главное - практику в написании своих приложений.

Здесь затрагивается много интересных тем, включая создание макетов приложений, работу с Email, SQLite, MediaPlayer, Vibration API, подключение сторонних библиотек и работу с сетью.

Андрей.

Prologistic.com.ua

Оглавление

1. Создание и начальная настройка проекта Android.

В каждом из наших 5 простых приложений мы будем использовать один шаблон проекта.

1. Создание проекта

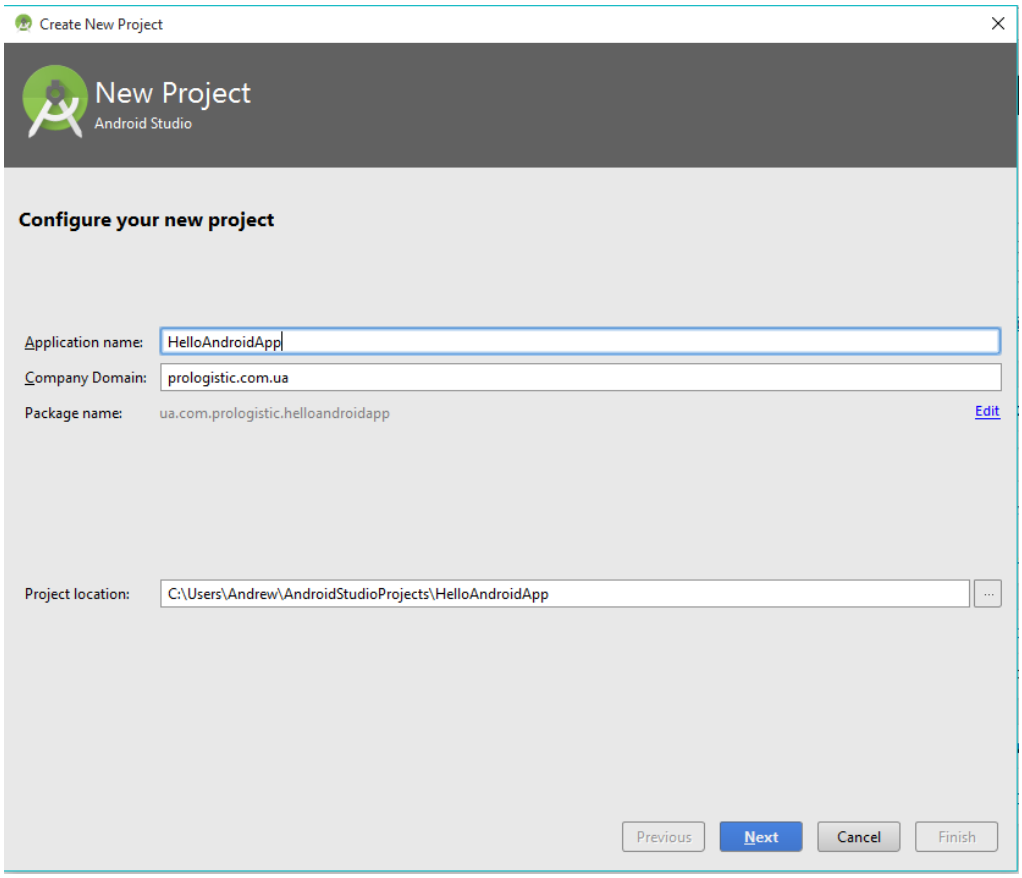
Если у Вас еще не установлен Android SDK, Java и IDE Android Studio, то обратитесь к этому пошаговому руководству.

Если среда разработки и Android SDK уже установлены, то Вы готовы к созданию проекта:

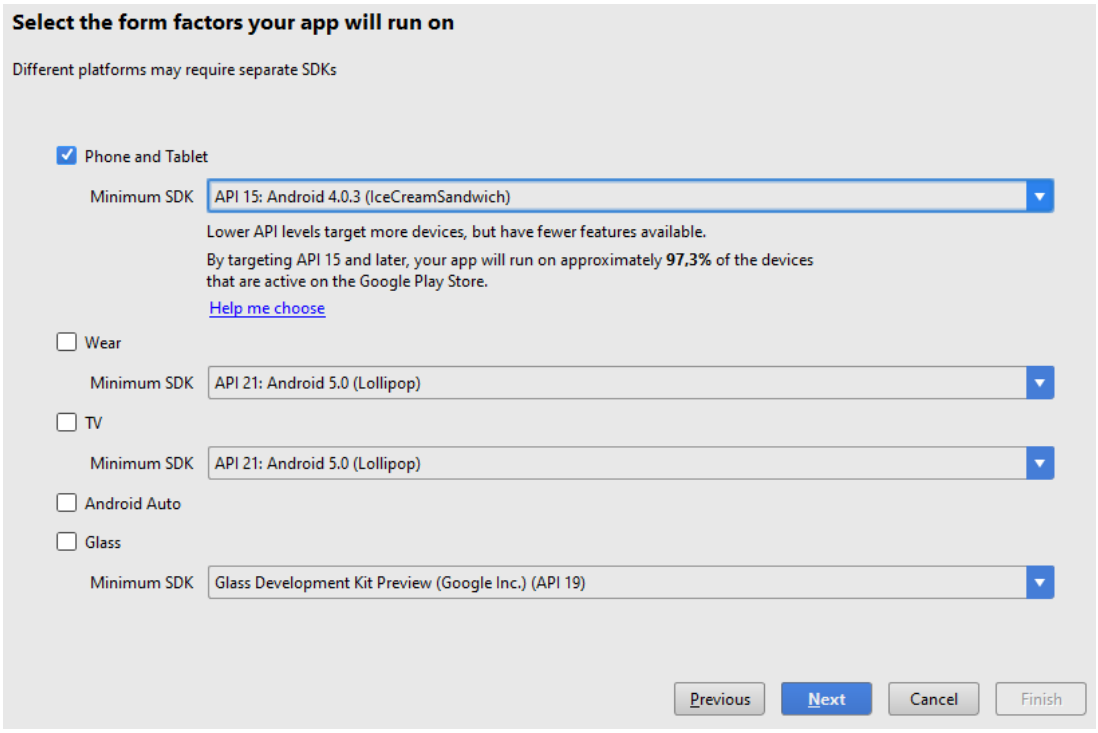
Запустите Android Studio. При запуске Android Studio в первый раз будет представлен экран приветствия, предлагая вам несколько вариантов для начала работы:

На экране приветствия видны варианты импорта проекта из системы контроля версий, Eclipse ADT, проекта Gradle или из готовых примеров кода, которые предоставляются вместе с Android. Из этого списка выберите пункт **Start a new Android Studio project**.

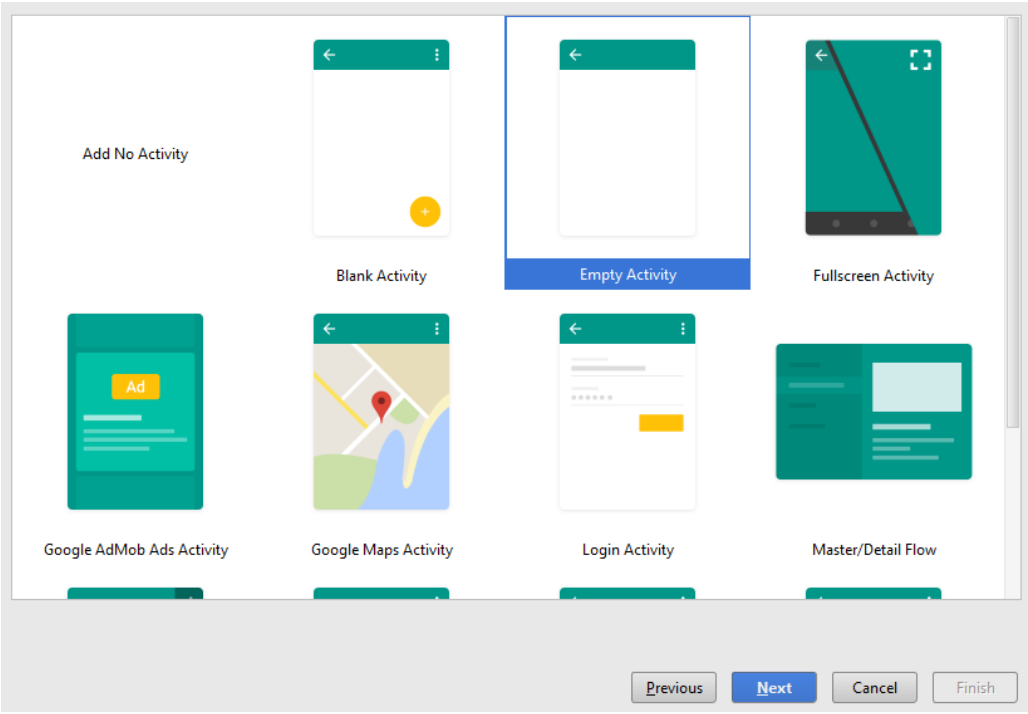
Далее нас просят ввести Название приложения (**Application name**) и Домен (**Company Domain**), который потом будет реверсирован в имя пакета. Вводим названия и нажимаем кнопку **Next**.



Далее нам нужно выбрать минимальную версию Android, для которого будет доступно наше приложение. Из всех доступных вариантов я бы предпочел выбирать Android 4.0 (API 15):

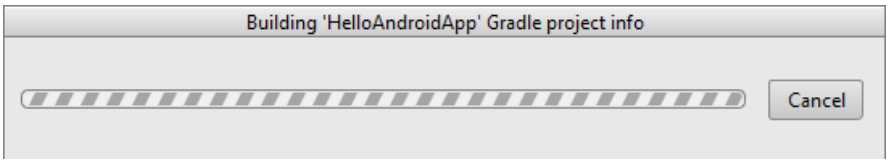


Жмем **Next** и попадаем на следующий экран, где нам будет представлен список возможных вариантов Hello world проектов с некоторыми уже настроенными компонентами. Нам нужен пункт **Empty Activity**:

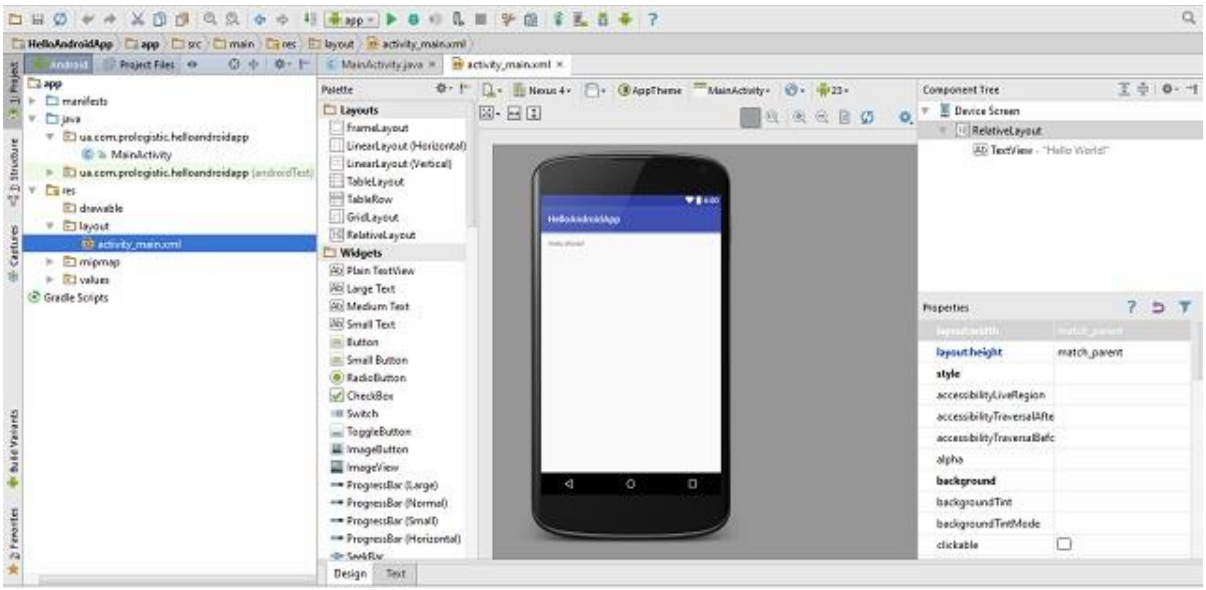


Нажимаем **Next** и в следующем экране **Finish** (ничего при этом не меняя).

После нажатия на кнопку **Finish**, у вас должно появиться маленькое окно с информацией о ходе генерации проекта:



Это может занять довольно много времени, но не больше 1-2 минуты.

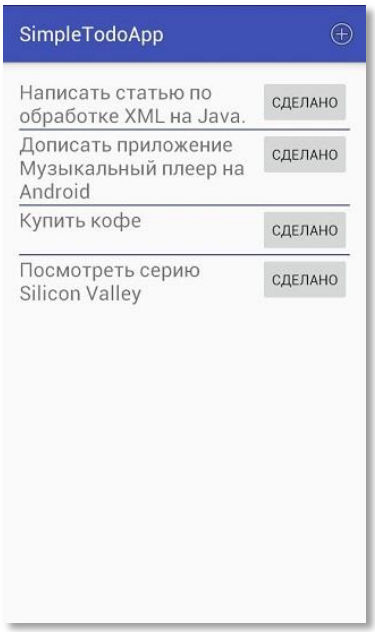


Все, проект был создан, а мы готовы к разработке первого простого приложения. В каждом новом разделе я буду ссылаться на этот пункт книги для создания нового проекта. Подробнее о структуре проекта можно узнать здесь.

Теперь можно переходить к разделу Практики.

2. Практика

2.1 Создание простого приложения «SimpleTodoApp»



2.1.1 Следуйте инструкциям пункта **1.1 Создание проекта** и создайте новый проект с Application Name (названием): «SimpleTodoApp», остальные пункты можете оставить без изменений, просто нажимая кнопку Next.

2.1.2 Краткое описание приложения

Todo App – классическое приложение для начинающих разработчиков. Чаще всего подобные приложения пишутся для получения навыков работы с базой данных SQLite и знакомством со списками.

2.1.3 Разработка

Создание макета приложения

Макет **activity_main.xml** будет содержать корневой элемент RelativeLayout и вложенный компонент ListView, который представляет собой список с заметками:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <ListView
        android:id="@+id/list_todo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:divider="@color/colorPrimary"
        android:dividerHeight="1dp" />
</RelativeLayout>
```

Обратите внимание, что виджет ListView содержит атрибут **android:divider**, который задает цвет разделителя и **android:dividerHeight**, который задает высоту разделителя. В данном случае **1dp** будет достаточно.

Для создания новых элементов списка нам потребуется отдельный макет **item.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center_vertical"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/title_item"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_toLeftOf="@+id/delete_item"
        android:layout_toStartOf="@+id/delete_item"
        android:textSize="20sp" />

    <Button
        android:id="@+id/delete_item"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:onClick="deleteItem"
        android:text="@string/deleteItemTitle" />

</RelativeLayout>
```

Обратите внимание, что мы определили прямой вызов метода **deleteItem** после нажатия на кнопку **delete_item**.

Также нам потребуется Меню с пунктом «Добавить новую заметку». Создайте новый файл **main.xml** в каталоге **/res/menu**:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/action_add_item"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/todoItemTitle"
        app:showAsAction="always" />
</menu>
```

Все, что связано с макетом мы создали, теперь приступим к реализации базы данных.

Работа с базой данных

Для сохранения заметок нам потребуется база данных лишь с одной таблицей, которая будет хранить саму заметку и id заметки.

Создайте подкаталог **database** в пакете приложения. Далее создайте в нем класс DBContract со следующим содержимым:

```
// выносим основные данные по таблице, столбцам и БД в т.н. класс-контракт
public class DBContract {
    public static final String DB_NAME =
"ua.com.prologistic.simpletodoapp.db";
    public static final int DB_VERSION = 1;

    public class TodoEntry implements BaseColumns {
        public static final String TABLE = "todo";
        public static final String COL_TITLE = "title";
    }
}
```

TaskContract класс определяет константы, которые используются для доступа к данным в базе данных. Также нам нужен вспомогательный класс DataBaseHelper создания самой базы данных. Создать этот класс в пакете **database** и добавьте следующий код:

```
public class DataBaseHelper extends SQLiteOpenHelper {

    public DataBaseHelper(Context context) {
        super(context, DBContract.DB_NAME, null, DBContract.DB_VERSION);
    }
         // метод для создания простой таблицы с 2 столбцами в БД
    @Override
    public void onCreate(SQLiteDatabase db) {
        // запрос на создание таблицы в БД
        String createTable = "CREATE TABLE " +
                DBContract.TodoEntry.TABLE + " ( " +
                DBContract.TodoEntry._ID +
                " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                DBContract.TodoEntry.COL_TITLE +
                " TEXT NOT NULL);";
        db.execSQL(createTable);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
        db.execSQL("DROP TABLE IF EXISTS " + DBContract.TodoEntry.TABLE);
        onCreate(db);
    }
}
```

Теперь идем в класс **MainActivity**, инициализируем меню и в методе **onOptionsItemSelected** определяем вызов метода **createAlertDialog()** для добавления новой заметки в БД:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return super.onCreateOptionsMenu(menu);
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    switch (id) {
        case R.id.action_add_item:
            createAlertDialog().show();
            return true;
        default: return super.onOptionsItemSelected(item);
    }
}
```

Теперь в том же **MainActivity** объявим метод **createAlertDialog()** и необходимые ему методы:

```
// создает диалог для добавления новой заметки
private AlertDialog createAlertDialog() {
    final EditText editTextItem = new EditText(this);
    return new AlertDialog.Builder(this)
            .setTitle("Добавление новой заметки")
            .setMessage("Введите текст заметки:")
            .setView(editTextItem)
            .setPositiveButton("Добавить",
                new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog1, int which) {
                        String item = String.valueOf(editTextItem.getText());
                        // добавляем новую запись в БД
                        insertIntoDB(item);
                        updateUI();
                    }
                })
            .setNegativeButton("Отмена", null)
            .create();
}
     // добавляет новую запись в БД
private void insertIntoDB(String item) {
    SQLiteDatabase db = mDbHelper.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(DBContract.TodoEntry.COL_TITLE, item);
    db.insertWithOnConflict(DBContract.TodoEntry.TABLE,
            null,
            values,
            SQLiteDatabase.CONFLICT_REPLACE);
    db.close();
}
```

Как видно из листинга, метод **createAlertDialog()** создает новый диалог с названием, сопутствующим сообщением и полем для ввода заметки. Также есть 2 кнопки: «Добавить», после нажатия на которую вызывается метод **insertIntoDb(String item)**, и кнопка «Отмена», которая просто убирает диалог с экрана пользователя.

Метод **insertIntoDb(String item)** получает в качестве параметра текст из поля ввода заметки и добавляет новую запись в таблицу БД.

Подробнее о создании, настройке и методах компонента AlertDialog можно почитать здесь.

Как Вы заметили, в коде сразу после метода **insertIntoDB(item)** есть вызов еще одного метода **updateUI()**. Этот метод используется для обновления интерфейса пользователя после удаления или добавления новой заметки. Дело в том, что интерфейс не знает о том, что в диалоге была создана и сохранена в базу данных новая заметка. Для того, чтобы пользователь всегда имел достоверные результаты, мы должны обновлять интерфейс после каждой операции удаления или добавления заметки.

Что происходит в методе **updateUI()**? Сначала мы создаем пустой список для заметок, потом делаем запрос в БД и получаем результирующий курсор со всем содержимым таблицы. Проходим по всем элементам и заполняем список заметок. Полученным списком инициализируем адаптер или переопределяем ранее созданный адаптер.

```
private void updateUI() {
    // список для существующих заметок
    List<String> itemsList = new ArrayList<>();
    SQLiteDatabase db = mDbHelper.getReadableDatabase();
    Cursor cursor = db.query(DBContract.TodoEntry.TABLE,
            new String[]{DBContract.TodoEntry._ID,
                    DBContract.TodoEntry.COL_TITLE},
            null, null, null, null, null);
    // считываем все заметки из таблицы в список
    while (cursor.moveToNext()) {
        int idx = cursor.getColumnIndex(DBContract.TodoEntry.COL_TITLE);
        itemsList.add(cursor.getString(idx));
    }
    // заполняем адаптер списком заметок из БД
    if (mTodoAdapter == null) {
        mTodoAdapter = new ArrayAdapter<>(this,
                R.layout.item, R.id.title_item, itemsList);
        mItemsListView.setAdapter(mTodoAdapter);
    } else {
```

```
            mTodoAdapter.clear();
            mTodoAdapter.addAll(itemsList);
            mTodoAdapter.notifyDataSetChanged();
        }
        cursor.close();
        db.close();
    }
```

Теперь нам нужно инициализировать сам адаптер, соединение с базой данных и ListView в методе **onCreate()**:

```
    // объект для работы с БД
private DataBaseHelper mDbHelper;

private ArrayAdapter<String> mTodoAdapter;
// объекты для работы с интерфейсом
private ListView mItemsListView;

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

    mDbHelper = new DataBaseHelper(this);
    mItemsListView = (ListView) findViewById(R.id.list_todo);

    updateUI();
}
```

Когда мы создавали XML макет для MainActivity, то объявили прямой вызов метода **deleteItem()** в виджете Button с помощью атрибута **onClick**. Теперь нам осталось объявить этот метод для удаления заметки:

```
// Это метод для удаления заметки
// он вызывается, когда пользователь нажимает на кнопку 'Сделано'
public void deleteItem(View view) {
    View parent = (View) view.getParent();
    TextView itemTextView = (TextView) parent.findViewById(R.id.title_item);
    String item = String.valueOf(itemTextView.getText());

    // удаляем заметку по тексту в TextView
    SQLiteDatabase db = mDbHelper.getWritableDatabase();
    db.delete(DBContract.TodoEntry.TABLE,
            DBContract.TodoEntry.COL_TITLE + " = ?",
            new String[]{item});
    db.close();
    // обновляем адаптер, отвечающий
    // за отображение заметок в интерфейсе
    updateUI();
}
```

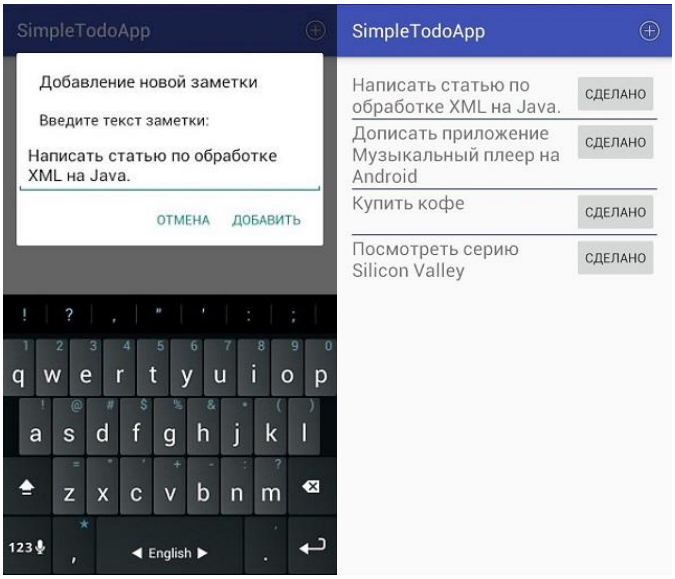
Как видно из листинга, мы получаем текст заметки и вызываем стандартный метод удаления строки из таблицы БД.

Теперь добавим в проект недостающие XML-файлы:

```
<resources>
    <string name="app_name">SimpleTodoApp</string>
    <string name="todoItemTitle">Добавить заметку</string>
    <string name="deleteItemTitle">Сделано</string>
</resources>
```

Файл **AndroidManifest.xml** не требует никаких модификаций, поэтому его здесь приводить не будем.

Результаты:



Вот и все, что нужно сделать для создания простого приложения «Список дел» или «Заметки». Оно работает с базой данных, имеет простейший интерфейс и является неплохим началом дальнейшей самостоятельной разработки под Android.

Скачать полный проект для Android Studio можно здесь.

2.2 Создание простого приложения «E-mail Client»

2.2.1 Настройка проекта

Следуйте инструкциям пункта **1.1 Создание проекта** и создайте новый проект с Application Name (названием): «SimpleEmailClient», остальные пункты можете оставить без изменений, просто нажимая кнопку Next.

2.2.2 Краткое описание приложения

Email Client – простое приложение для отправки писем. В нем мы рассмотрим два способа отправить письмо в Android: с помощью библиотеки JavaMail и стандартного клиента Android.

2.2.3 Разработка

Макет приложения

В **activity_main.xml** макете у нас будет 3 поля для ввода данных (Имей получателя, Тема письма и Текст письма), подписи к ним и 2 кнопки для отправки с помощью JavaMail и Email клиента Android:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/emailToText" />
    <EditText
        android:id="@+id/emailTo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textEmailAddress" />
    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/emailSubject" />
    <EditText
        android:id="@+id/emailSubject"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textEmailSubject" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/emailText" />
    <EditText
        android:id="@+id/emailMessage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/messageHintText"
        android:lines="5" />
    <Button
        android:id="@+id/btnJavaMailSend"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/javamailText" />
    <Button
        android:id="@+id/btnIntentSend"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/androidEmailText" />
</LinearLayout>
```

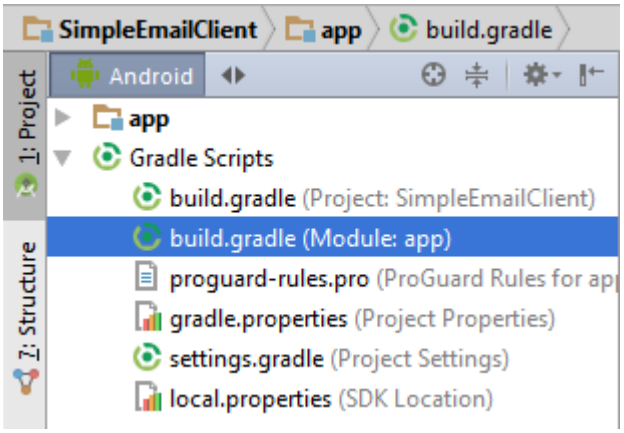
Также нам нужно дать нашему приложению разрешение на использование Интернета в файле AndroidManifest.xml:

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

Работа с Email

Настройка среды

Чтобы использовать библиотеку JavaMail в приложении на Android, нам нужно добавить его в зависимости проекта. Так как мы используем Android Studio, то за сборку проекта отвечает Gradle. Находим в структуре проекта файл **build.gradle (Module: app)**:

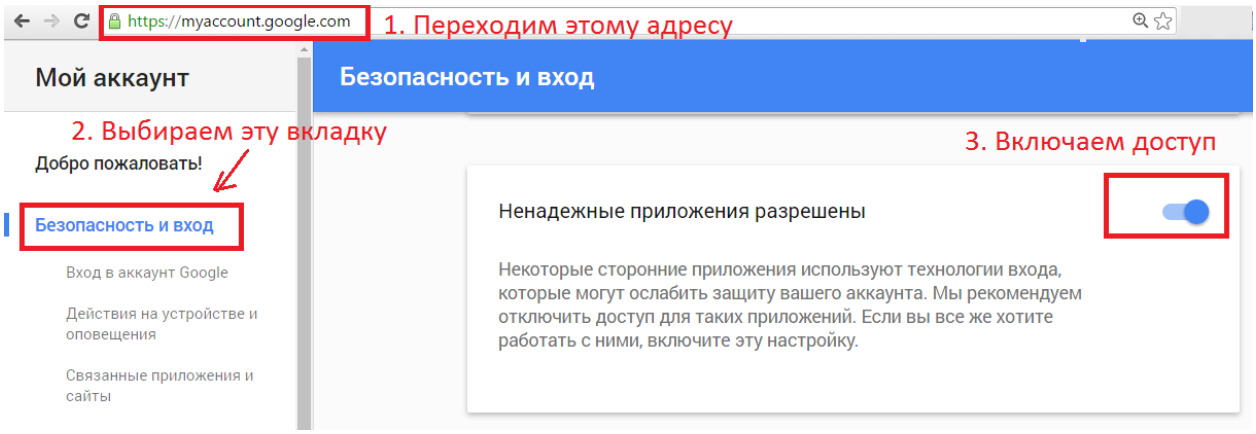


Открываем его и в разделе **dependencies** добавляем две библиотеки, отвечающие за отправку сообщений с помощью JavaMail специально на Android:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.2.1'
    compile 'com.sun.mail:android-mail:1.5.5'
    compile 'com.sun.mail:android-activation:1.5.5'
}
```

(если Вы хотите добавить эти библиотеки в папку libs, то Вам нужно их скачать здесь)

Также нам потребуется разрешить доступ к Gmail почте через smtp протокол в настройках аккаунта. Для этого войдите в свою учетную запись Google и включите доступ для таких приложений:



Если Вы не хотите предоставлять доступ, то переходите к следующему пункту, где мы отправляем письмо с помощью стандартного приложения Android.

Код для работы с почтой

Для работы с почтой у нас будет 2 класса:

В первом классе EmailConfig будет два поля: EMAIL и PASSWORD отправителя. Я решил захардкодить эти данные в коде для простоты приложения. Вы можете изменить это добавив еще два поля в наше приложение с email адрессом и паролем отправителя.

Листинг класса ниже:

```
public class EmailConfig {
    public static final String EMAIL ="ВАШ ИМЕЙЛ";
    public static final String PASSWORD ="ВАШ ПАРОЛЬ";
}
```

Также нам потребуется класс для отправки писем SendEmail. Он будет наследован от класса AsyncTask, чтобы отправка письма проходила в фоновом потоке. Также нам потребуется виджет ProgressDialog для отображения прогресса отправки письма:

```
// Наследумся от AsyncTask, чтобы отправка письма проходила в фоновом потоке
public class SendEmail extends AsyncTask<Void,Void,Void> {
    private Context context;
    private Session session;
    private String email;
    private String subject;
    private String message;
    //виджет Progressdialog для отображения прогресса отправки письма
    private ProgressDialog progressDialog;

    public SendEmail(Context context, String email, String subject, String
message){
        this.context = context;
        this.email = email;
        this.subject = subject;
        this.message = message;
    }
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressDialog = new ProgressDialog(context);
        progressDialog.setTitle("Отправляем письмо...");
        progressDialog.show();
    }
    @Override
    protected void onPostExecute(Void aVoid) {
        super.onPostExecute(aVoid);
        // убираем диалог с экрана после завершения отправки сообщения
        progressDialog.dismiss();
        // оповещаем пользователя об успешной отправке письма
        Toast.makeText(context,"Письмо отправлено",Toast.LENGTH_LONG).show();
    }
    @Override
    protected Void doInBackground(Void... params) {
        // конфигурационные данные для GMail
        Properties props = new Properties();
        props.put("mail.smtp.host", "smtp.gmail.com");
        props.put("mail.smtp.socketFactory.port", "465");
        props.put("mail.smtp.socketFactory.class",
"javax.net.ssl.SSLSocketFactory");
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.port", "465");

        // создаем соединение с почтовым сервером GMail
        session = Session.getDefaultInstance(props,
                new javax.mail.Authenticator() {
                    // проходим авторизацию в почтовый акаунт
                    protected PasswordAuthentication
getPasswordAuthentication() {
                        return new PasswordAuthentication(EmailConfig.EMAIL,
```

```
EmailConfig.PASSWORD);
                    }
                });
        try {
            // Создаем объект MimeMessage для компоновки письма
            MimeMessage mMessage = new MimeMessage(session);

            // устанавливаем адресс отправителя
            mMessage.setFrom(new InternetAddress(EmailConfig.EMAIL));
            // устанавливаем адресс получателя
            mMessage.addRecipient(Message.RecipientType.TO, new
InternetAddress(email));
            // устанавливаем тему письма
            mMessage.setSubject(subject);
            // устанавливаем текст письма
            mMessage.setText(message);

            // отправляем сообщение
            Transport.send(mMessage);

        } catch (MessagingException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

Как видно из листинга, в методе **onPreExecute()** мы создаем диалог для отображения прогресса отправки письма. В методе **doInBackground()** мы создаем соединение с почтовым сервером GMail, проходим авторизацию в почтовый аккаунт, создаем компоновку письма и отправляем письмо.

Обратите внимание, что если вы хотите отправить письмо не с помощью аккаунта GMail, то Вам нужно будет найти host и порты для желаемого почтового сервера в Интернете.

Теперь в классе MainActivity нам нужно инициализировать объявленные в макете виджеты и назначить слушатели:

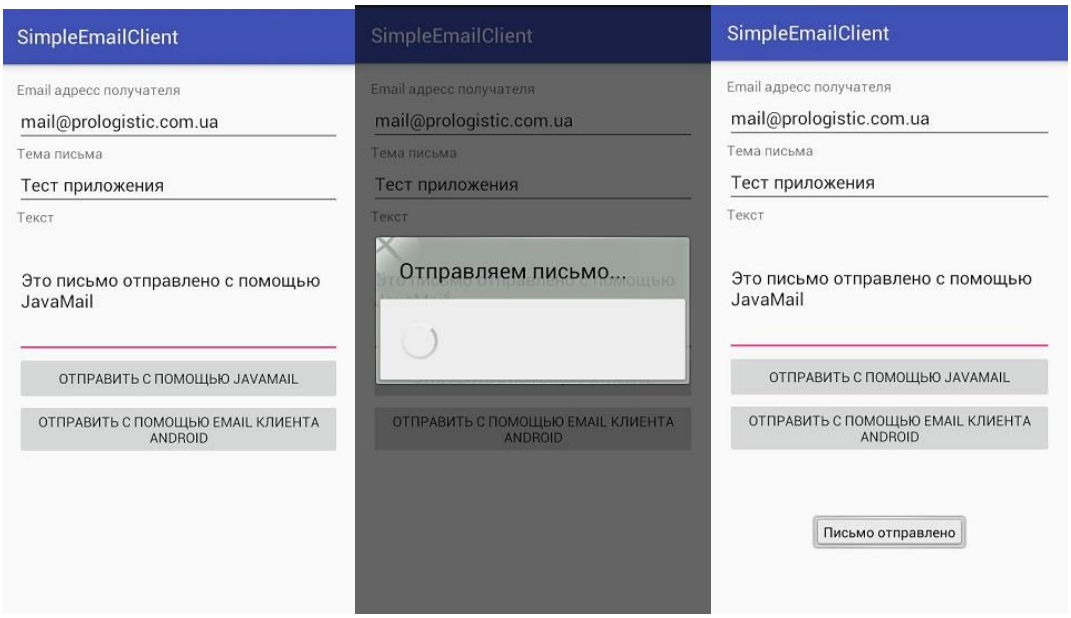
```
public class MainActivity extends AppCompatActivity implements
View.OnClickListener {
    private EditText editTextEmailTo;
    private EditText editTextSubject;
    private EditText editTextMessage;
    // кнопки для отправки сообщений с помощью JavaMail
    private Button btnJavaMail;
    // и стандартных клиентов Android
    private Button btnWithIntent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // инициализируем виджеты
        editTextEmailTo = (EditText) findViewById(R.id.emailTo);
        editTextSubject = (EditText) findViewById(R.id.emailSubject);
```

```
        editTextMessage = (EditText) findViewById(R.id.emailMessage);
        btnJavaMail = (Button) findViewById(R.id.btnJavaMailSend);
        btnWithIntent = (Button) findViewById(R.id.btnIntentSend);
        //Adding click listener
        btnJavaMail.setOnClickListener(this);
        btnWithIntent.setOnClickListener(this);
    }
    private void sendEmail() {
        // получаем введенные данные
        String email = editTextEmailTo.getText().toString().trim();
        String subject = editTextSubject.getText().toString().trim();
        String message = editTextMessage.getText().toString().trim();
        // получаем объект для отправки имейла с помощью JavaMail
        SendEmail sm = new SendEmail(this, email, subject, message);
        // отправляем имейл
        sm.execute();
    }
    @Override
    public void onClick(View v) {
        int id = v.getId();
        switch (id) {
            case R.id.btnJavaMailSend:
                sendEmail();
                break;
            case R.id.btnIntentSend:
                sendEmailWithDefaultClient();
                break;
        }
    }
    private void sendEmailWithDefaultClient() {
        // получаем введенные данные
        String emailTo = editTextEmailTo.getText().toString().trim();
        String subject = editTextSubject.getText().toString().trim();
        String message = editTextMessage.getText().toString().trim();

        Intent email = new Intent(Intent.ACTION_SEND);
        email.putExtra(Intent.EXTRA_EMAIL, new String[]{ emailTo});
        email.putExtra(Intent.EXTRA_SUBJECT, subject);
        email.putExtra(Intent.EXTRA_TEXT, message);
        // указываем, что нам нужны только почтовые клиенты
        email.setType("message/rfc822");
        startActivity(Intent.createChooser(email, "Выберите Email клиент
:"));
    }
}
```

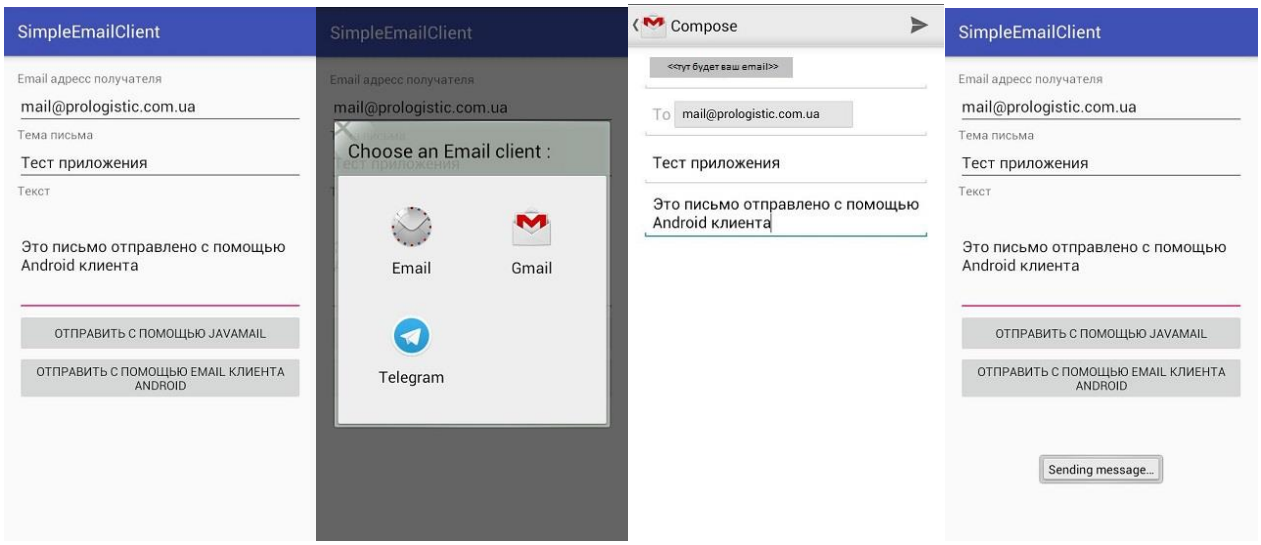
Как видно из кода выше, мы используем два способа отправки писем: метод **sendEmail()** используется для отправки письма с помощью JavaMail, а в методе **sendEmailWithDefaultClient()** мы используем Intent с action **Intent.ACTION_SEND** (это значит, что мы хотим отослать что-то) с типом **message/rfc822** (в данном случае используется для фильтрации только по почтовым клиентам). После вызова этого метода будет создан диалог с предложением выбрать желаемый клиент для завершения действия.

Результаты. Отправляем с помощью JavaMail



1. Вводим данные 2. Происходит отправка в фоне 3. Сообщение об успешной отправке

Отправляем с помощью клиента Android:



1. Вводим данные 2. Выбираем Gmail клиент 3. Попадаем в Gmail клиент 4. Отправляется

[Скачать проект приложения можно здесь](#).

2.3 Создание простого приложения «Screen Crack App»

2.3.1 Настройка проекта

Следуйте инструкциям пункта **1.1 Создание проекта** и создайте новый проект с Application Name (названием): «ScreenCrackApp», остальные пункты можете оставить без изменений, просто нажимая кнопку Next.

2.3.2 Краткое описание приложения

ScreenCrackApp – шуточное приложение для развлечения. В нем мы предлагаем пользователю поучаствовать в соревновании, в ходе которого он должен успеть нажать на кнопку более 100 раз всего за 20 секунд. Но как бы пользователь не старался успеть, его всегда ожидает «треснутый экран» смартфона в результате усердных нажатий на кнопку.

2.3.3 Разработка

Макет приложения

В макете **activity_main.xml** мы разместим всего 3 виджета: 2 виджета Button и виджет TextView для отображения сообщения. Первая кнопка будет перезапускать игру, а на вторую пользователь должен будет нажимать 100 раз (на самом деле меньше ☺). В виджете TextView нас будет предложение начать игру:

```
    <?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:id="@+id/background">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/titleText"
        android:textStyle="bold"
        android:textSize="20sp"
        android:gravity="center"
        android:id="@+id/title"
```

```
        android:layout_below="@+id/btnRetry"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="25dp"
        android:text="@string/initText"
        android:layout_gravity="center"
        android:id="@+id/btnRun"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/retryText"
        android:id="@+id/btnRetry"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Теперь нам нужно подготовить ресурсы для приложения. Нам понадобится картинка с битым экраном и звук битого стекла. Найти картинку и звук Вы можете самостоятельно или воспользоваться теми, что нашел я (в конце раздела есть ссылка на скачивание проекта со всеми ресурсами).

*Если Вы решили найти сами, то звуковой файл нужно разместить в папке **res/raw/** (на папке **res** нажать правой кнопкой мыши и выбрать пункт **New -> Android Resource Directory**. В появившемся окне в поле выбрать **Resource type: raw**). Картинку битого стекла вставьте в папку **res/drawable/**.*

Далее нам нужно добавить разрешение на доступ к функции вибрации телефона. Добавьте в файле **AndroidManifest.xml** следующее разрешение:

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

Работа с MainActivity

В приложении нам понадобятся два виджета Button: для перезапуска игры и для отсчета кликов пользователя. Также мы будем использовать объект MediaPlayer для проигрывания звука битого стекла и объект Vibrator для вибрирования после «треска экрана» телефона. Объект RelativeLayout будет использоваться для установки фона треснутого стекла. Теперь у нас есть все для создания нужных эффектов:

```
public class MainActivity extends AppCompatActivity implements
View.OnClickListener {
    private MediaPlayer mPlayer;
    private Button btnRun;
    private Button btnRetry;
    private Vibrator mVibrator;
    private RelativeLayout relativeLayout;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // инициализируем объект MediaPlayer для проигрывания
            // звука из папки res/raw/
        mPlayer = MediaPlayer.create(this, R.raw.screencracksound);
        // инициализируем объект Vibrator для вибрирования во время нажатия
        mVibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
        btnRun = (Button) findViewById(R.id.btnRun);
        btnRetry = (Button) findViewById(R.id.btnRetry);
        relativeLayout = (RelativeLayout) findViewById(R.id.background);
        btnRun.setOnClickListener(this);
        btnRetry.setOnClickListener(this);
    }
    public void onClick(View v) {
        int id = v.getId();
        switch (id) {
            case R.id.btnRun: {
                // получаем текущее число из кнопки
                int counter = Integer.parseInt(btnRun.getText().toString());
                // уменьшаем значение счетчика по нажатию на кнопку
                btnRun.setText(--counter);
                // создаем объект Random для генерации
                     // случайного числа в диапазоне
                Random random = new Random();
                // если генерированное число равно 10 или меньше 50
                if ((random.nextInt(10) + 1 == 10) || (counter < 50)) {
                    // "разбиваем" стекло экрана
                    doCrack();
                } break;
            }
            // если нажата кнопка заново,
            // обнуляем значения виджетов интерфейса
            case R.id.btnRetry: {
                // очищаем фон макета
                relativeLayout.setBackgroundResource(0);
                // делаем кнопку счетчика видимой
                btnRun.setVisibility(View.VISIBLE);
                // устанавливаем началое значение 100 на кнопке-счетчике
                btnRun.setText(String.valueOf(100));
                break;
            }
        }
    }
    private void doCrack() {
    // вызываем вибрацию
    doVibration();
    // начинает проигрывание файла
    playMusic();
    // устанавливаем фон треснутого стекла
    setBackground();
}
```

```
        private void playMusic() {
            // начинает проигрывание файла
            mPlayer.start();
        }
        // устанавливаем изображение битого стекла на фон
        private void setBackground() {
            relativeLayout.setBackgroundResource(R.drawable.crack);
            // делаем кнопку счетчика невидимой
            btnRun.setVisibility(View.GONE);
        }
        private void doVibration() {
            // Вибрируем на протяжении полусекунды (0.5 сек = 500 миллисек)
            long[] pattern = {0, 500, 0};
            mVibrator.vibrate(pattern, -1);
        }
        protected void onDestroy() {
            super.onDestroy();
            // освобождаем объект MediaPlayer после уничтожения объекта Activity
            if (mPlayer != null) {
                mPlayer.release();
                mPlayer = null;
            }
        }
}
```

Как видно из листинга, у нас есть метод **onClick**, в котором мы управляем двумя кнопками. Если пользователя нажал **R.id.btnRun** (кнопка обратного отсчета), то у нас генерируется случайное число – если оно попадает в указанный диапазон, то экран «разбивается», вызывается вибрация и появляется звук битого стекла. Если пользователь нажал кнопку **R.id.btnRetry**, то экран нашего шуточного приложения обновляется и пользователя может начать заново.

Результат:

Вот такое простое приложения затронуло работу с API MediaPlayer, Vibration и взаимодействие с RelativeLayout.

Скачать проект с исходным кодом приложения можно здесь.

Практический мини курс «Разработка 3 простых приложения на Android» подошел к концу.

Курс затронул множество интересных тем, подробнее с которыми Вы можете ознакомиться в официальной документации Android или на сайте http://prologistic.com.ua, где регулярно публикуются новые статьи по Java и разработке под Android.