ЛАБОРАТОРНАЯ РАБОТА № 3

(задание – в конце методички)

Цель лабораторной работы:

Разработка простого приложения, помогающего понять структуру приложения, освоить основные *операторы*, привыкнуть к среде разработки.

Задачи лабораторной работы:

- создать новое приложение и изучить его структуру;
- настроить интерфейс приложения;
- реализовать логику приложения.

4.1 Введение

Для достижения поставленной цели в лабораторной работе создадим *приложение* в среде разработки *Android IDE* (Eclipse и *ADT*), подробно рассмотрим структуру полученного проекта и разберем назначение основных его элементов.

Чтобы дальнейшие действия приобрели некоторый смысл, сформулируем задачу, которую будет решать наше *приложение*, назовем его "Угадай число". Суть приложения в том, что *программа* случайным образом "загадывает" число от 0 до 100, а *пользователь* должен угадать это число. При каждом вводе числа, *программа* сообщает пользователю результат: введенное число больше загаданного, меньше или же "Ура, победа!" число угадано.

Разрабатываемое *приложение* выполняет свои функции только когда видимо на экране, когда оно не видимо его работа приостанавливается, т. е. имеем дело с приложением переднего плана. Для выполнения всей работы достаточно определить одну *активность* в приложении, фоновые процессы не предусмотрены.

Далее в работе рассмотрим простейшие элементы интерфейса пользователя и добавим их в *приложение*, а также рассмотрим вопросы, связанные непосредственно с программированием: научимся обрабатывать события, возникающие при взаимодействии приложения с пользователем; реализуем логику проверки числа на совпадение с загаданным.

4.2 Создание приложения и изучение его структуры

Создайте новый проект в среде *Android IDE* (Eclipse с *ADT*). Процесс создания нового проекта и описание основных настроек подробно рассмотрен в лабораторной работе к первой лекции.

В процессе создания проекта, мы назвали его ProjectN, *среда разработки* подготавливает необходимые папки и файлы. Полный иерархический *список* обязательных элементов проекта можно увидеть на вкладке *Package* Explorer (аналогичную информацию предоставляет вкладка *Project*Explorer), *иерархия* полученных папок и файлов для нашего проекта изображена на рис. 1.

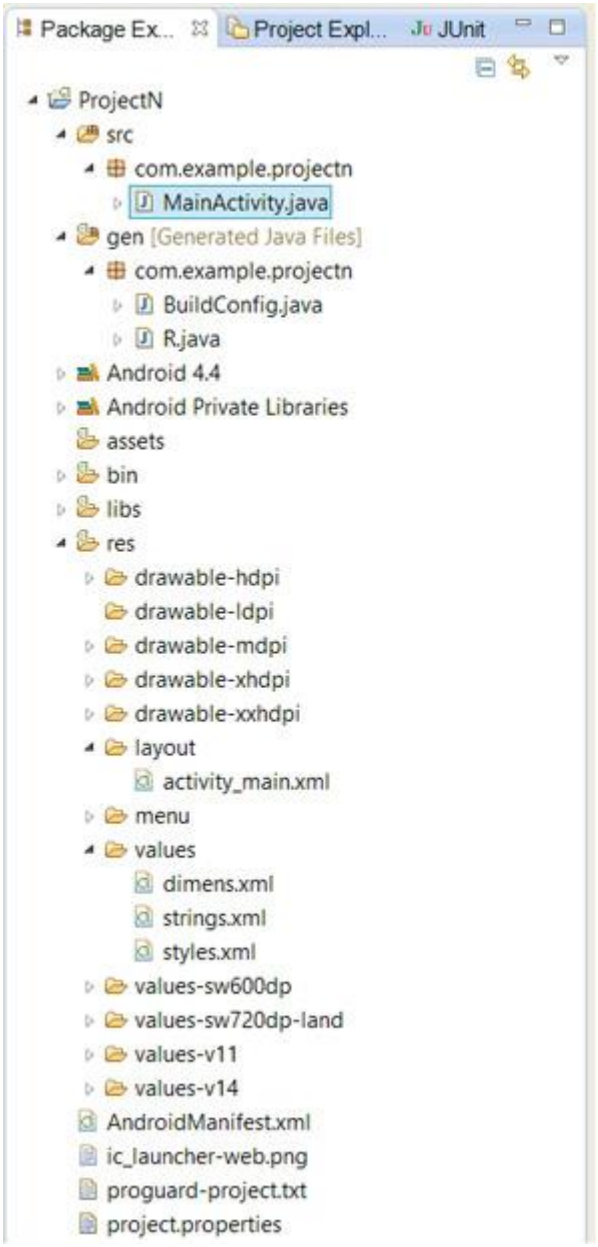

Рис. 1. Структура проекта ProjectN

В настоящее время нас будет интересовать назначение нескольких файлов и папок.

Рассмотрим папки:

- папка **src** - содержит файлы с исходным кодом на языке Java. Именно в этой папке размещаются все классы, создаваемые в процессе разработки приложения. Сейчас в этой папке в пакете com.example.projectn размещается единственный класс `MainActivity.java`. Этот класс определяет главную и единственную активность в этом приложении.

Комментарий 1: Имя пакету присваивается в процессе создания приложения в *поле* **Package Name**, использовать *com.example* не рекомендуется, т. к. пакет с таким именем нельзя загрузить в Google *Play*. Часто рекомендуют использовать в качестве имени пакета название сайта программиста, записанное в обратном порядке, можно просто использовать свои имя и фамилию. Последнее *слово* в имени пакета формируется автоматически и совпадает с именем проекта.

Комментарий 2: Имя файлу присваивается в процессе создания приложения на этапе настройки активности. Имя определяется в *поле***Activity Name**.

Комментарий 3: *Package* Explorer отображает структуру папок, которая создается в каталоге, выбранном в качестве рабочего (Workspace) при запуске Eclipse. Например, рабочий каталог называется workspaceADT, в нем для нашего проекта появилась *папка* с именем ProjectN, в ней есть*папка* src, в ней *com*, в ней *example*, в ней projectn (заметьте, что название пакета распалось на три папки, каждое *слово*, отделенное точкой определило новую папку). И уже в папке projectn находится *файл* MainActivity.*java* и будут размещаться другие *java*-файлы проекта.

- папка **gen** - содержит java-файлы, которые не требуется изменять и лучше вообще не трогать. Эти файлы генерируются автоматически. Нас может заинтересовать файл R.java он содержит идентификаторы (ID) для всех ресурсов приложения.
- папка **res** - содержит структуру папок ресурсов приложения, рассмотрим некоторые из них:
  - **layout** - в данной папке содержатся xml-файлы, которые описывают внешний вид форм и их элементов, пока там находится только activity_main.xml;
  - **values** - содержит XML файлы, которые определяют простые значения, таких ресурсов как, строки, числа, цвета, темы, стили, которые можно использовать в данном проекте;
  - **menu** - содержит XML файлы, которые определяют все меню приложения.

Рассмотрим *файл* **AndroidManifest.xml** - *файл* в формате *xml*, который описывает основные свойства проекта, разрешение на использование ресурсов устройства и др. Сразу после создания приложения *файл* AndroidManifest.*xml* выглядит так, как показано на рис. 2.

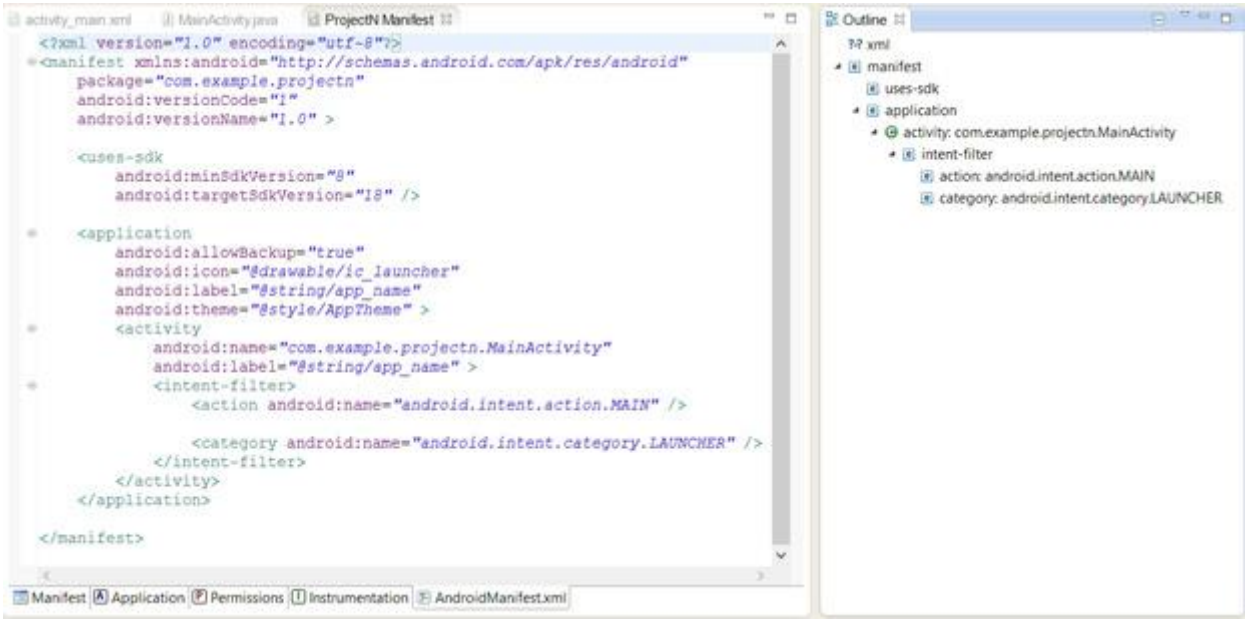

Рис. 2. Файл AndroidManifest.xml только созданного проекта

Рассмотрим подробно *файл* манифеста.

Первый обязательный элемент `<manifest>` является корневым элементом файла, должен содержать обязательный элемент `<application>`и все остальные элементы по необходимости. Рассмотрим основные атрибуты этого элемента:

`xmlns:android`	- определяет пространство имен Android, всегда должен иметь значение: "`http://schemas.android.com/apk/res/android`". Обязательный атрибут.
`package`	- полное имя пакета, в котором располагается приложение. Обязательный атрибут. Имя должно быть уникальным, может содержать заглавные и строчные латинские буквы, числа и символ подчеркивания. Однако начинаться должно только с буквы. Для избежания конфликтов с другими

	разработчиками рекомендуется использовать имя вашего сайта (если он есть) записанное в обратном порядке. В нашем случае пакет имеет имя "com.example.projectn" и наше приложение не удастся разместить в Google Play (но мы на это и не претендуем).**Внимание:** если Вы опубликовали свое приложение, Вы **не можете менять имя пакета**, т.к. имя пакета служит уникальным идентификатором для приложения и в случае его смены приложение будет рассматриваться, как совсем другое и пользователи предыдущей версии не смогут его обновить.
`android:versionCode`	- внутренний номер версии приложения не виден пользователю. Этот номер используется только для определения является ли одна версия более современной по сравнению с другой, больший номер показывает более позднюю версию.
`android:versionNumber`	- номер версии, является строкой и используется только для того, чтобы показать пользователю номер версии приложения.
`android:shareUserID`, `android:sharedUserLabel`, `android:installLocation`	- эти атрибуты в нашем файле манифеста не представлены, про их назначение можно почитать по ссылке: http://developer.android.com/guide/topics/manifest/manifest-element.html.

Рассмотрим элемент `<uses-sdk>`, который показывает совместимость приложения с версиями *Android*. Основные атрибуты:

`android:minSdkVersion`	- указывает значение минимального уровня API, необходимого для работы приложения. Система Android не позволит установить приложение, если уровень API ниже, чем уровень, указанный в этом атрибуте. **Внимание:**если этот атрибут не указан, система установит значение по умолчанию равным "1", которое означает, что приложение совместимо со всеми версиями Android. И в случае, если приложение не совместимо со всеми версиями, установка пройдет на любую версию Android, а во время работы приложение сломается, когда попытается обратиться к недоступным элементам API. Поэтому необходимо всегда указывать значение этого атрибута.
`android:targetSdkVersion`	- указывает уровень API целевой платформы Android приложения, если этот атрибут пропущен, по умолчанию принимается значение `android:minSdkVersion`.
`android:maxSdkVersion`	- указывает максимальное значение уровня API, под который разрабатывалось приложение. Если значение этого атрибута ниже, чем уровень API соответствующий версии Android, на которую устанавливается приложение, то система не позволит установить такое приложение.

Если внимательно посмотреть на *файл* манифеста рис. 2, можно заметить, что данный *атрибут* в нем отсутствует. На самом деле разработчикам не рекомендуются задавать *значение* этого атрибута. Во-первых, это *значение* будет препятствовать использованию приложений на новых версиях *Android* при их появлении, несмотря на то, что все новые версии полностью обратно-совместимы. Во-вторых, стоит иметь ввиду, что задание этого атрибута может привести к тому, что *приложение* будет удалено с устройства после обновления *Android*до более высокого уровня *API*.

Подробнее с элементом <uses-sdk> и его атрибутами можно ознакомиться по ссылке: http://developer.android.com/guide/topics/manifest/uses-sdk-element.html.

Рассмотрим элемент <application>, который является обязательным элементом манифеста, полностью определяет состав приложения. Представляет собой *контейнер* для элементов <activity>, <service>, <receiver>, <provider> (и не только), каждый из которых определяет соответствующий *компонент* приложения. Содержит набор атрибутов, действие которых распространяется на все компоненты приложения. Рассмотрим атрибуты элемента <application>, представленные в манифесте на рис. 4.2:

android:allowBackup - определяет разрешение для приложения участвовать в резервном копировании и восстановлении. Если значение этого атрибута false, то для приложения никогда не может быть создана резервная копия, даже если проводится резервное копирование всей системы целиком. По умолчанию значение этого атрибута равно true.

android:icon - определяет иконку для приложения целиком, а также иконку по умолчанию для компонентов приложения, которая может быть переопределена атрибутом android:icon каждого компонента. Задается как ссылка на графический ресурс, содержащий изображение, в нашем случае значение этого атрибута равно "@drawable/ic_launcher".

android:label - определяет видимый для пользователя заголовок приложения целиком, а также заголовок по умолчанию для компонентов приложения, который может быть переопределен атрибутом android:label каждого компонента. Задается как ссылка на строковый ресурс, в нашем случае значение атрибута равно "@string/app_name".

android:theme - определяет тему по умолчанию для всех активностей приложения, может быть переопределен атрибутом android:theme каждой активности. Задается как ссылка на стилевой ресурс, в нашем случае значение атрибута равно"@style/AppTheme".

На самом деле у элемента <application> гораздо больше атрибутов, чем нам удалось рассмотреть, найти полный *список* атрибутов с описаниями можно по ссылке: http://developer.android.com/guide/topics/manifest/application-element.html.

В приложении всего одна *активность*, других компонентов нет, в связи с этим элемент **<application>** в манифесте содержит ровно один элемент <*activity*> и больше никаких других элементов не содержит. Рассмотрим элемент <activity>, который определяет *активность*. Для каждой активности обязательно необходим свой элемент <activity> в манифесте. Рассмотрим атрибуты элемента <activity>, представленные в манифесте на рис. 3:

android:name - определяет имя класса, который задает активность. Значение атрибута должно полностью определять имя класса с указанием пакета, в котором располагается класс. В нашем случае атрибут имеет значение:"com.example.projectn.MainActivity". Можно использовать сокращенную запись ".MainActivity", в этом случае добавляется имя пакета, определенное соответствующим атрибутом элемента <manifest>.

`android:configChanges` - перечисляет изменения конфигурации, которыми может управлять активность. Если конфигурация меняется во время работы, то по умолчанию активность останавливается и перезапускается. Если же изменение конфигурации указано в этом атрибуте, то при появлении этого изменения активность не перезапускается, вместо этого она продолжает работать и вызывает метод `onConfigurationChanged()`. В нашем случае атрибут имеет значение`"orientation|keyboardHidden|screenSize"`, т. е. при смене ориентации экрана, смене размера экрана и изменении доступности клавиатуры не произойдет перезапуск активности.

`android:label` - определяет видимый пользователю заголовок активности, если он отличается от общего заголовка приложения. Задается как ссылка на строковый ресурс, в нашем случае значение атрибута равно `"@string/app_name"`(т.е. можно было и не задавать).

`android:theme` - определяет тему активности, если она отличается от общей темы приложения, заданной соответствующим атрибутом элемента `<application>`. Задается как ссылка на стилевой ресурс, в нашем случае значение атрибута равно `"@style/FullscreenTheme"`.

На самом деле у элемента `<activity>` гораздо больше атрибутов, чем нам удалось рассмотреть, найти полный *список* атрибутов с описаниями можно по ссылке: http://developer.android.com/guide/topics/manifest/application-element.html.

В манифесте для нашего приложения элемент `<activity>` содержит ровно один элемент: `<intent-filter>`, определяющий типы намерений, которые может принимать *активность*. Этот элемент содержит два элемента: `<action> и <category>`.

Первый элемент определяет действия, которые проходят в фильтр намерений, при этом `<intent-filter>` должен содержать хотя бы один элемент `<action>`, в противном случае ни один *объект*-намерение не сможет пройти через фильтр и *активность* не возможно будет запустить. Элемент `<action>` имеет единственный *атрибут* `android:name="android.intent.action.MAIN"`.

Второй элемент определяет имя категории в фильтре намерений. Имеет единственный *атрибут*`android:name="android.intent.category.LAUNCHER"`.

На этом разбор манифеста приложения закончим, подробно с описанием всех элементов этого файла можно познакомиться по ссылке:http://developer.android.com/guide/topics/manifest/manifest-intro.html.

Чаще всего, при создании приложения приходится иметь дело с папками **src**, **res/layout** и **res/values**, т.к. там находятся основные файлы проекта.

4.3 Настройка интерфейса приложения

До того, как начнем формировать *интерфейс*, имеет смысл подготовить возможность проверки разрабатываемого приложения на ошибки. Чтобы не загружать каждый раз *приложение* на реальное устройство, в *Android SDK* предусмотрена возможность использования виртуального устройства (*AVD* или *Android virtual device*), эмулирующего работу реального смартфона. Процесс создания виртуального устройства или эмулятора подробно расписан в "Установка и настройка среды программирования ADT Bundle" .

Пришло время задуматься о внешнем виде приложения. Для начала необходимо определить какие элементы графического интерфейса нам нужны, как эти элементы будут располагаться на форме и каким образом будет реализовано взаимодействие с пользователем.

Так как *приложение* очень простое, то и *интерфейс* особой сложностью отличаться не будет. Нам потребуется *поле* для ввода чисел (**TextEdit**), текстовая *метка* для вывода информации (**TextView**) и кнопка для подтверждения введенного числа (**Button**). Располагать элементы интерфейса будем друг под другом, сверху информационная часть, ниже *поле ввода*, кнопку разместим в самом низу приложения. Взаимодействие приложения с пользователем организуется очень просто: *пользователь* вводит число в *поле* для ввода и нажимает кнопку, читает результат в информационном *поле* и, либо радуется победе, либо вводит новое число.

Если *пользователь* вводит правильное число, то *приложение* предлагает ему сыграть снова при этом кнопка будет играть роль подтверждения, а в информационное *поле* будет выведено приглашение к повторной игре.

Схематично *интерфейс* приложения изображен на рис. 3.

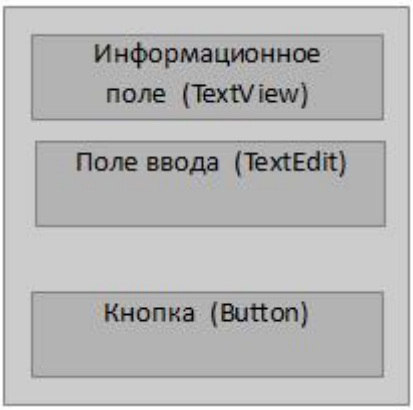

Рис. 3. Схема интерфейса приложения "Угадай число"

Нам необходимо добавить на форму три элемента: информационное *поле* (**TextView**), *поле ввода* (**TextEdit**) и кнопку (**Button**).

Android IDE поддерживает два способа для выполнения действий по формированию интерфейса приложения: первый основан на *XML*-разметке, второй относится к визуальному программированию и позволяет перетаскивать объекты интерфейса и размещать их на форме с помощью мыши. Считается, что визуальный способ подходит для новичков, а более продвинутые разработчики могут писать код вручную, однако чаще всего используется комбинированный подход.

Для формирования интерфейса будем работать с файлом **res/layout/activity_main.xml**. На рис. 4 можно увидеть редактор, соответствующий визуальному способу формирования интерфейса, этому режиму соответствует вкладка **Graphical Layout**.

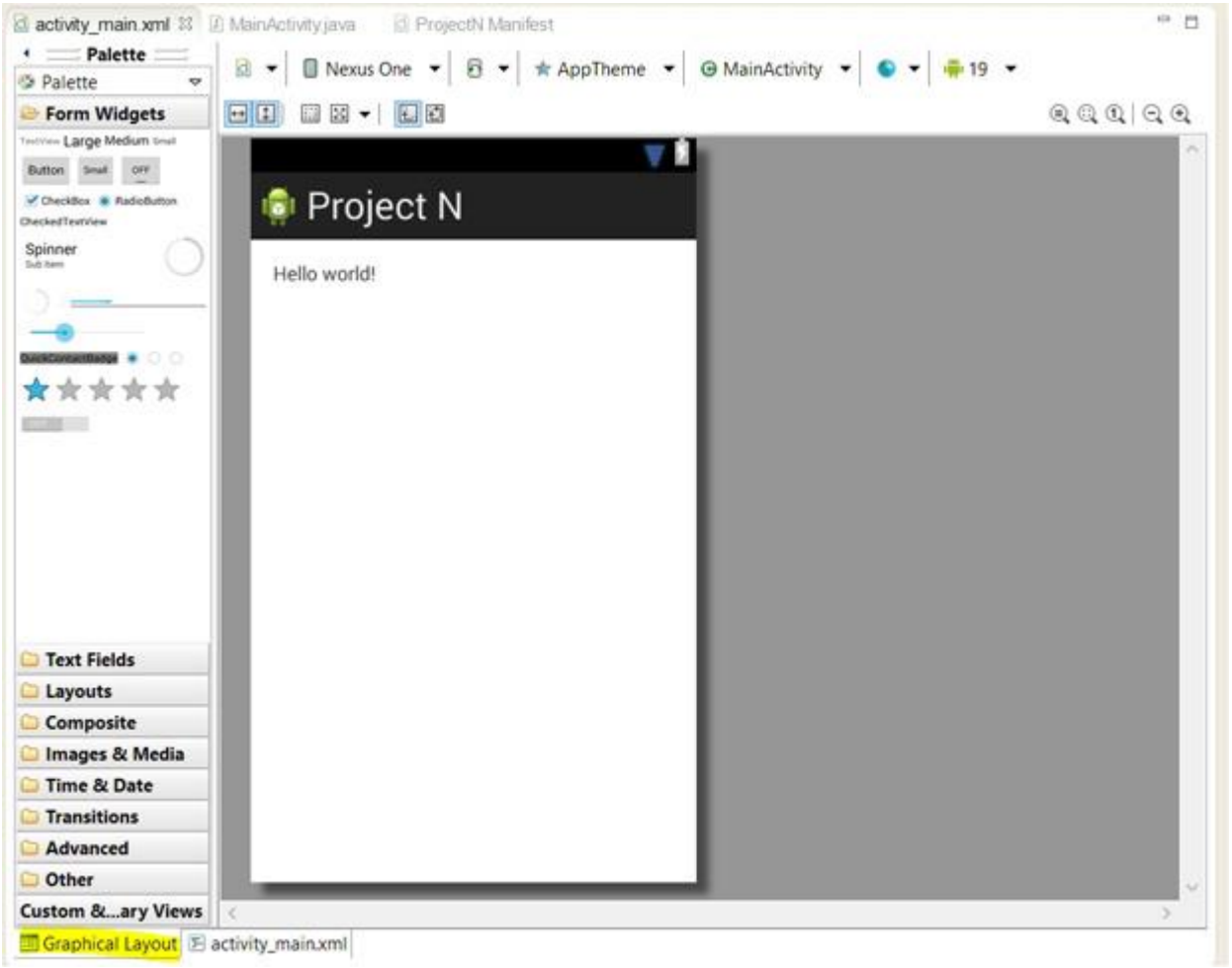

Рис. 4. Графическое изображение активности приложения

На рис. 4 рядом с вкладкой **Graphical Layout** расположена вкладка **activity_fullscreen.xml**. Она соответствует режиму редактирования интерфейса путем формирования *XML* файла. На рис. 5 можно увидеть редактор *XML* файла.

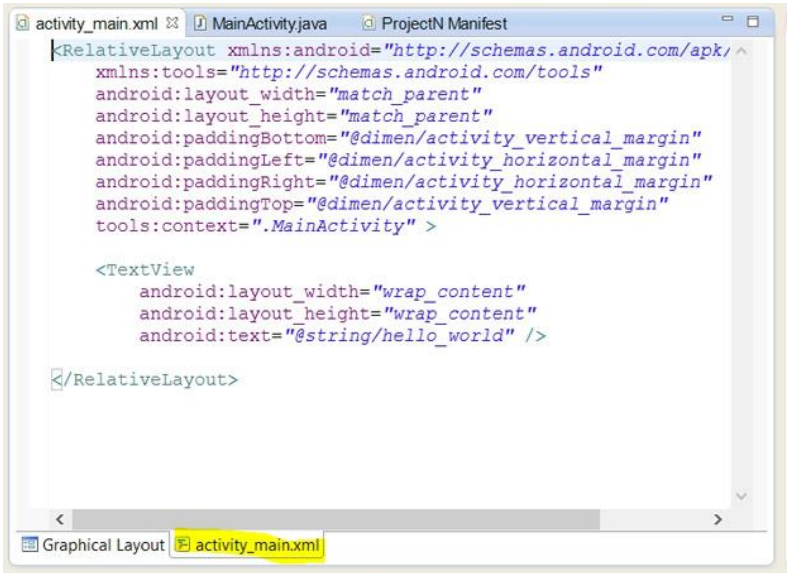

Рис. 5. Описание активности в XML формате

Зададим табличное расположение компонентов на форме, для этого выберем вкладку **Layouts**, найдем там **TableLayout** и добавим его на форму. На рис. 6 можно увидеть результат этих действий.

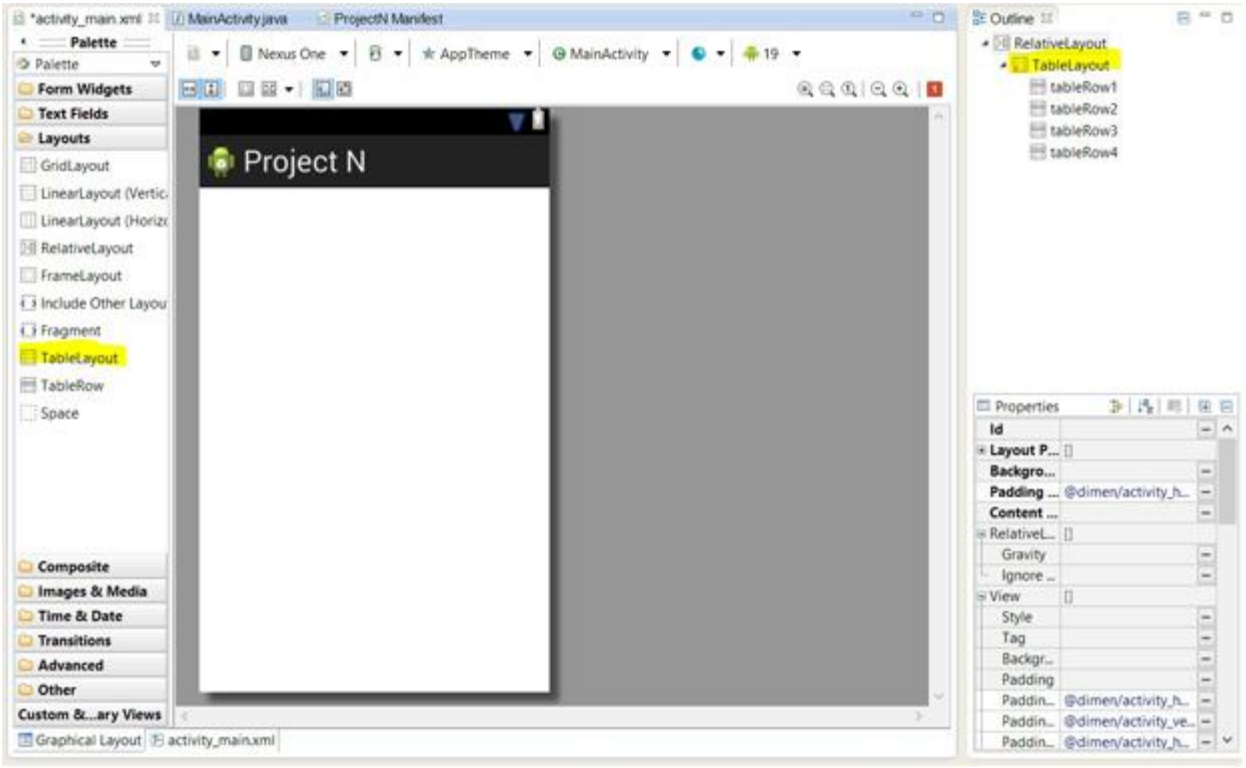

Рис. 6. Настройка интерфейса, добавление TableLayout

Теперь начнем добавлять элементы интерфейса, будем использовать *графический режим* правки.

Во-первых, нам необходимо добавить информационное *поле*. Для этого на панели **=Palette=** выбираем вкладку **Form Widgets**, на этой вкладке найдем *поле* **TextView**, перенесем в окно приложения, разместим в первой строке таблицы (**TableLayout**).

Во-вторых, нам потребуется *поле ввода* информации, на вкладке **Text Fields** найдем текстовое *поле* **Number** и разместим во второй строке таблицы.

В-третьих, вернемся на вкладку **Form Widgets**, выберем там элемент **Button** и добавим в третью строку таблицы. Не нужную четвертую строку таблицы удалим, получим следующий вид приложения, см. рис. 7.

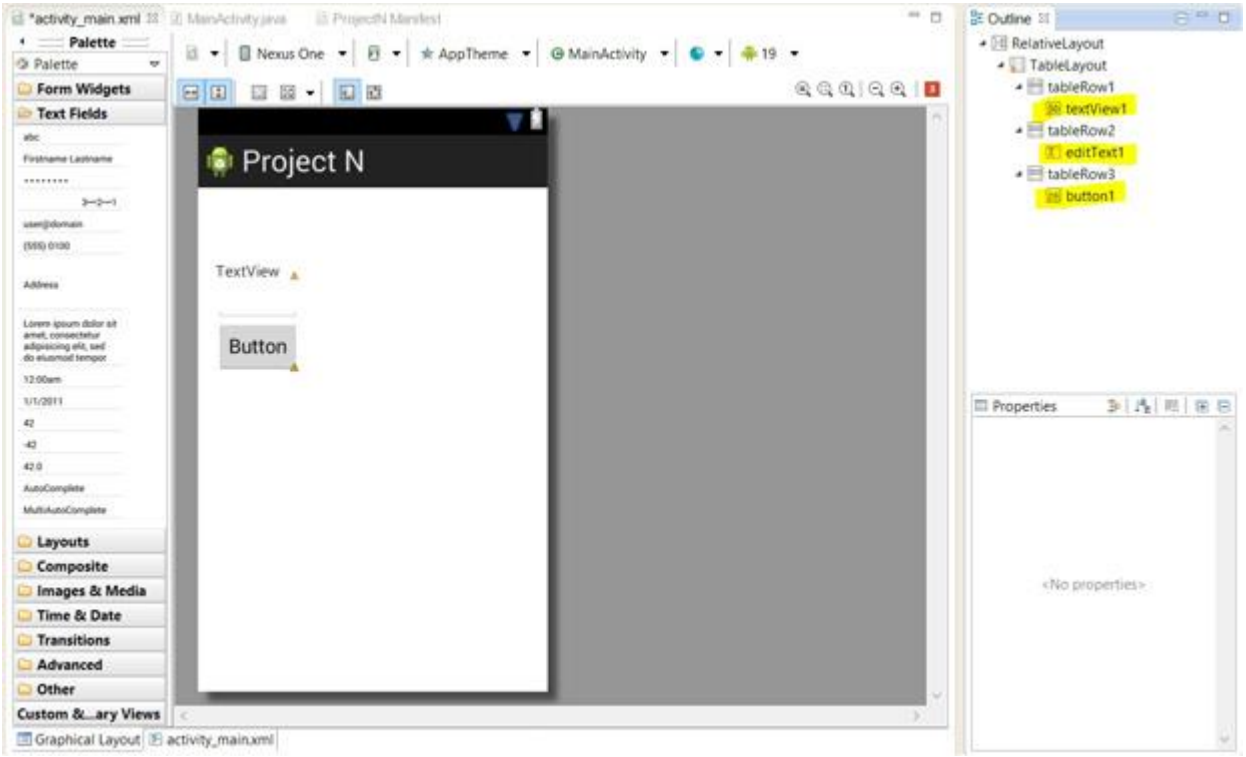

Рис. 4.7. Интерфейс приложения

После настройки интерфейса можно заглянуть в *файл* **activity_main.xml**, в этом файле прописано, что используется **TableLayout** и дано описание каждой из трех строк. На рис. 8 можно увидеть, как выглядит такое описание на примере первой строки таблицы.

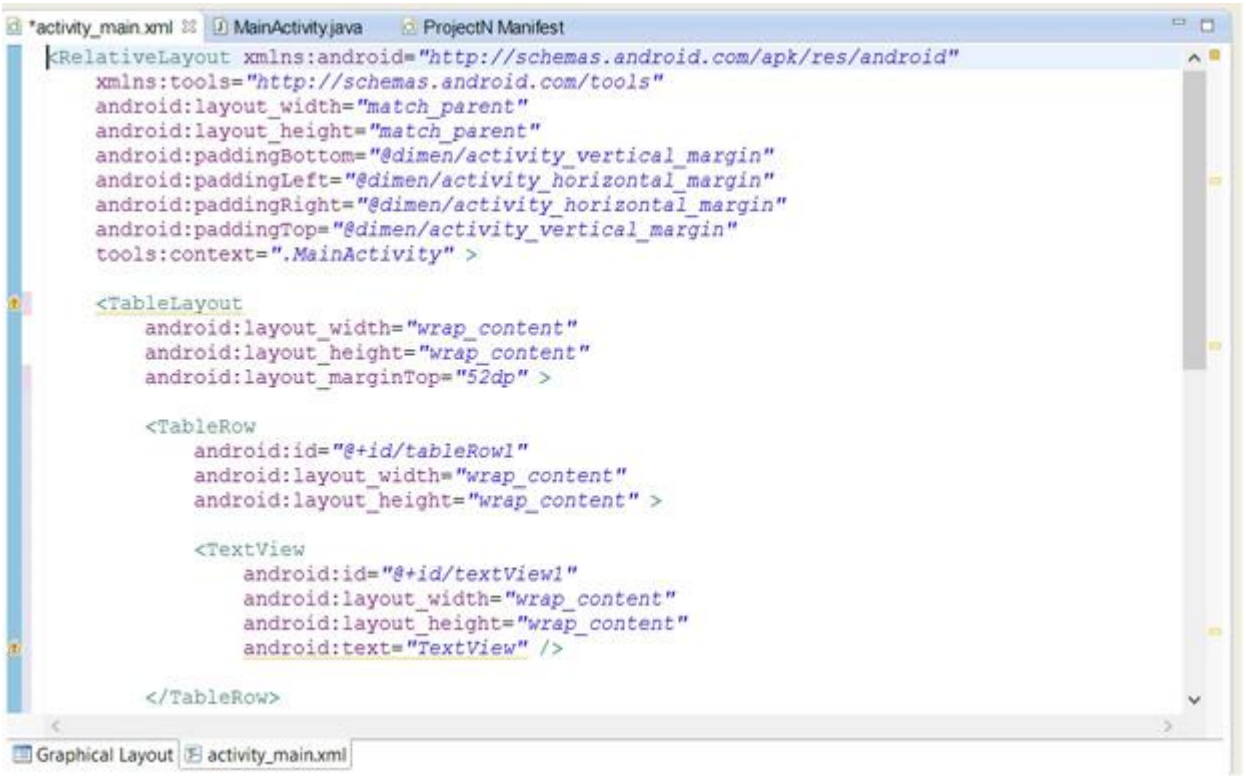

Рис. 8. Фрагмент файла activity_fullscreen.xml, описание строки в TableLayout

Теперь необходимо наполнить наши элементы интерфейса смыслом, нам понадобится текст для общения с пользователем, при программировании под *Android* существует практика разделять ресурсы и код приложения. Для хранения любых строк, которые могут понадобиться приложению, используется *файл* **strings.xml**. Хранение всех строковых ресурсов в этом файле серьезно облегчает

локализацию приложения на другие языки. Этот *файл* можно найти в *Package* Explorer в папке **res/values**. Откроем его и посмотрим, что там есть, см. рис. 9.

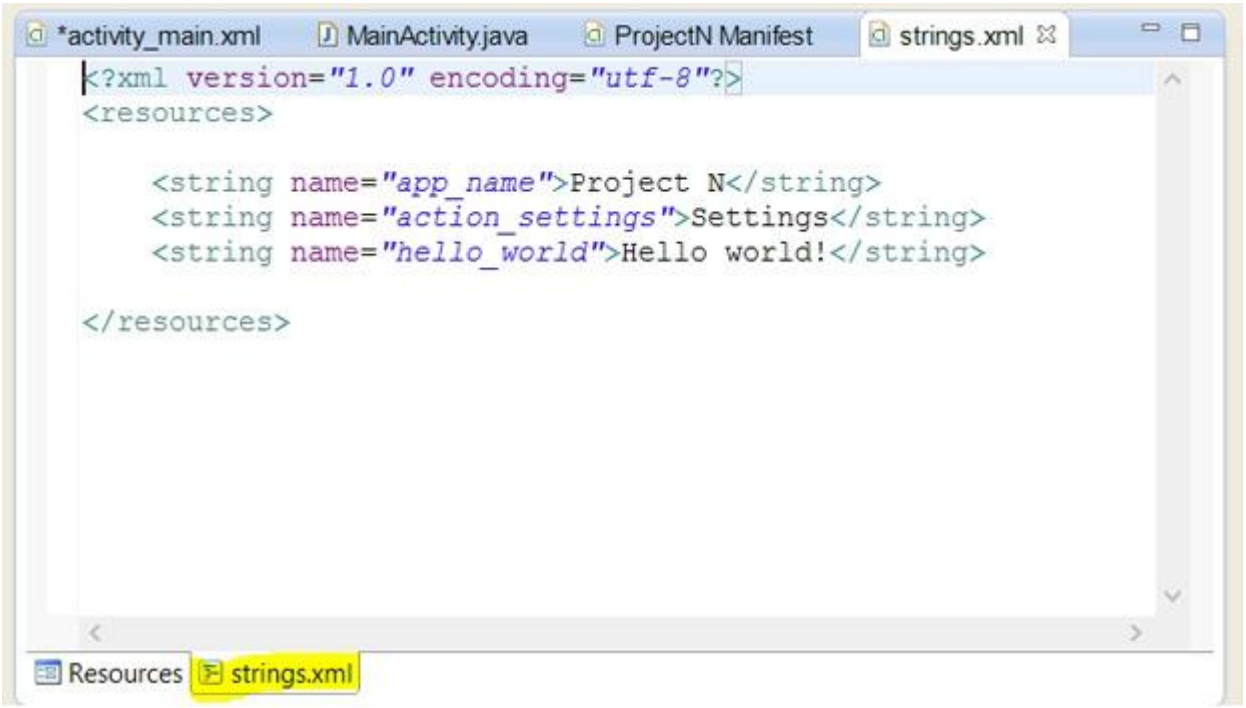

Рис. 9. Файл string.xml

Уберем лишние строки и добавим новые, результат можно посмотреть на рис. 10.

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

    <string name="app_name">Угадай число!</string>
    <string name="behind">Недолёт!</string>
    <string name="ahead">Перелёт!</string>
    <string name="hit">В точку!</string>
    <string name="input_value">Ввести значение</string>
    <string name="play_more">Сыграть ещё</string>
    <string name="try_to_guess">Попробуйте угадать число (1 &amp;#8211; 100) </string>
    <string name="error">Неверный ввод!</string>

</resources>
```

Рис. 10. Отредактированный файл string.xml

Данные переменные будут выполнять следующие задачи:

- `app_name` установит "видимое" название приложения;
- `behind`, `ahead`, `hit` оповестят пользователя об его успехах в игре;
- `play_more` и `try_to_guess` установит название кнопки, которое объяснит её функции;
- `input_value` пригласит пользователя к вводу числа;
- `error` сообщит о неверном вводе.

После изменения **strings.xml**, при переходе на другую вкладку, не забудьте сохранить изменения (самый быстрый способ - нажать **Ctrl+S**).

Настроим текст в информационном *поле*. Для этого на вкладке **Properties** в правой части окна выберем элемент **textView1** (это и есть наше информационное *поле*, имеет смысл придумать ему более осмысленное имя). Найдем свойство **Text**, подставим в него *значение* строки с именем `try_to_guess`, см. рис. 11.

Аналогично можно настроить текст, которым нас будет приветствовать кнопка, только в этом случае надо работать с элементом **button1**.

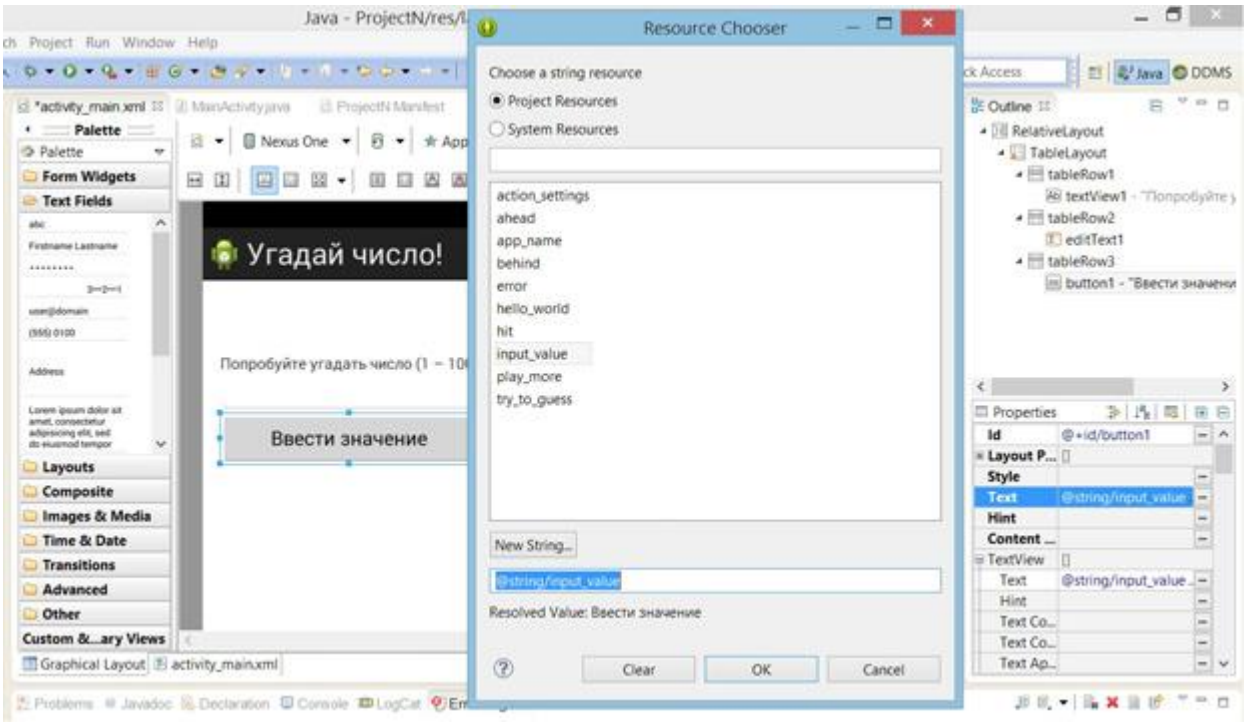

Рис. 11. Настройка текста для кнопки button1

Пришло время вспомнить о виртуальном устройстве, если оно работает, уже можно запустить проект и посмотреть, как *приложение* будет выглядеть на экране устройства, а выглядеть оно может как показано на рис. 12.

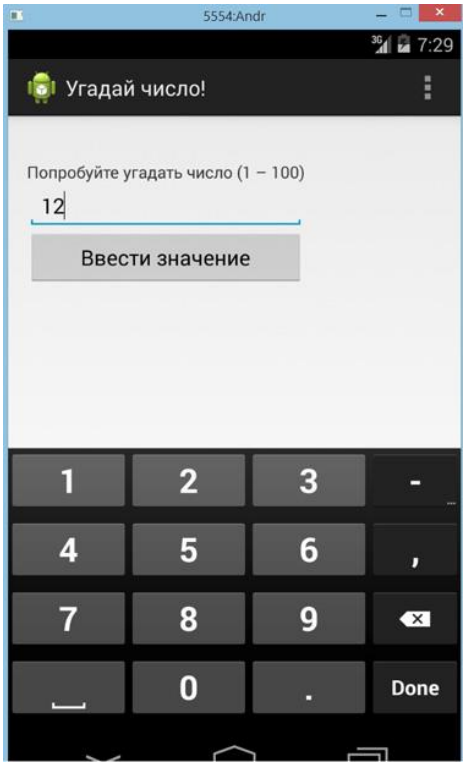

Рис. 12. Запуск приложения на виртуальном устройстве

Приложение выглядит довольно просто, но мы на многое и не рассчитывали. Главное, что нас интересует, это наличие всех элементов на экране, верный текст в каждом элементе, где он предусмотрен и возможность

вводить числа в *поле ввода*. На рис. 12 видно, что все требования выполнены. *Приложение* есть, его можно запустить на виртуальном или реальном устройстве, но оно ничего не делает. Следующим шагом будет реализация логики приложения, т. е. обработка события нажатия на кнопку, как было прописано в задании.

4 Реализация логики приложения

Приступим непосредственно к программированию, работать будем с файлом **src/com.example.projectn/MainActivity.java**. Найдем этот *файл* в *Package* Explorer см. рис. 13, откроем и начнем редактировать.

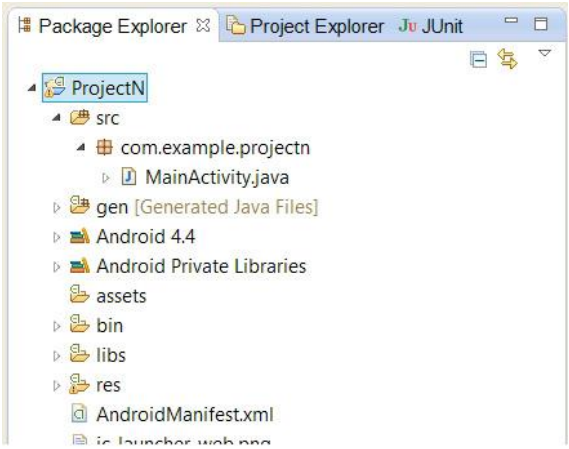

Рис. 13. Файл MainActivity.java в Package Explorer

Пока *файл* выглядит следующим образом, см. рис. 14.

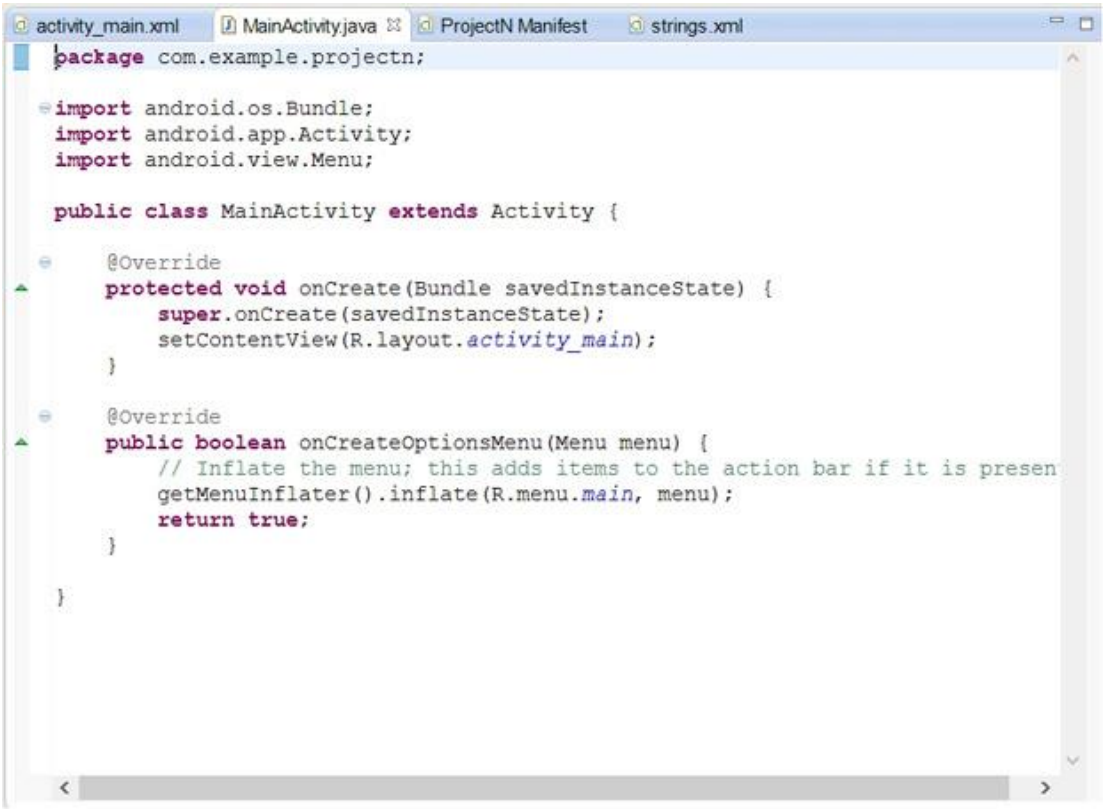

Рис. 14. Файл MainActivity.java после создания приложения

Можно заметить, что *класс* `MainActivity` является наследником класса `Activity` и в нем уже реализован метод `onCreate()`, который запускается при первоначальном создании активности, нам потребуется его дополнить, но об этом чуть позже.

Мы предполагаем программно менять информацию в *поле* **TextView**, получать *значение* из поля **EditText** и обрабатывать события нажатия на кнопку **Button**, поэтому необходимо объявить соответствующие переменные, как *поля класса* `MainActivity`:

```
TextView tvInfo;
EditText etInput;
Button bControl;
```

Чтобы не было ошибок, необходимо импортировать пакет *android.widget*, который содержит все элементы графического интерфейса:

```
import android.widget.*;
```

На самом деле *среда разработки* подскажет, что делать.

Теперь необходимо связать эти переменные с элементами интерфейса, уже добавленными нами в **activity_main.xml**, сделать это необходимо в методе `onCreate()`, а для получения уже созданного элемента интерфейса воспользуемся методом `findViewById()`. Итак в метод`onCreate()` добавим следующие строки:

```
tvInfo = (TextView)findViewById(R.id.textView1);
etInput = (EditText)findViewById(R.id.editText1);
bControl = (Button)findViewById(R.id.button1);
```

Метод `findViewById()` возвращает *объект* класса `View`, который является общим предком для всех компонентов пользовательского интерфейса, для того чтобы избежать возможных ошибок в скобках перед вызовом метода указываем до какого конкретно компонента необходимо сузить возможности объекта `View`.

Пришло время выполнить обработку нажатия на кнопку. Вернемся к файлу **activity_main.xml** в *графический режим* редактирования, выберем элемент **Button** и на вкладке со свойствами элемента найдем свойство **On Click** и запишем в него **onClick** - имя метода, который будет обрабатывать нажатие на кнопку. Как это выглядит показывает рис. 15.

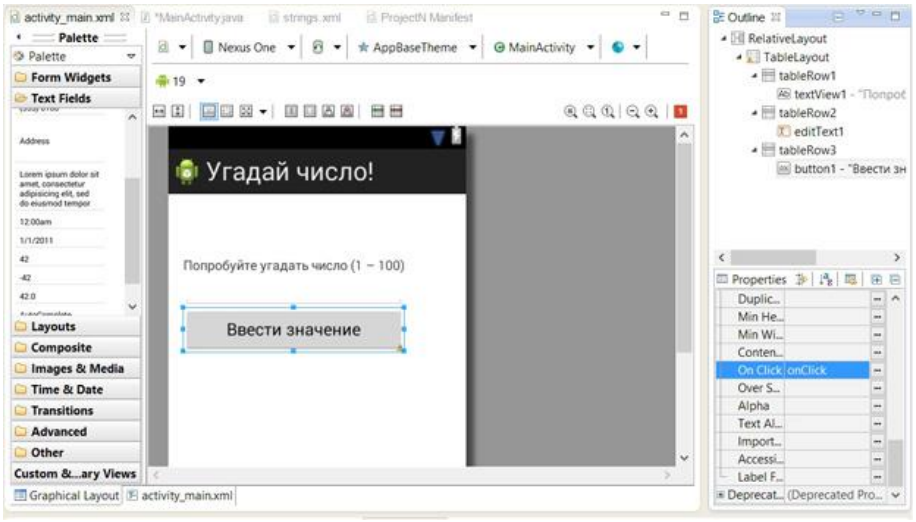

Рис. 15. Настройка свойства On Click для кнопки

Эти же действия можно выполнить в фале **activity_main.xml**, достаточно дописать выделенную на рис. 16 строку:

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="@string/input_value" />
```

Рис. 16. Настройка свойствами On Click для кнопки в файле XML

Для настройки свойств элементов интерфейса достаточно использовать любой способ: графический или редактирование *XML* файла.

Вернемся в *файл* **MainActivity.java**, в *класс* активности необходимо добавить метод:

```
public void onClick(View v){...}
```

Имя метода не обязательно должно быть `onClick()`, главное, чтобы оно совпадало с именем, указанным в свойстве **On Click**. В этом методе и будет происходить все наше *программирование* в этой лабораторной работе.

Нам потребуются две переменные:

- целочисленная для хранения загаданного числа (случайное число от 1 до 100);
- логическая для хранения состояния закончена игра или нет.

Обе эти переменные имеет смысл объявить как *поля класса* активности, первоначальные значения присвоить в методе `onCreate`.

Получить целочисленное *значение* из поля ввода, можно с помощью следующей конструкции:

```
Integer.parseInt(etInput.getText().toString())
```

изменить *значение* текста в информационном *поле* можно с помощью следующей конструкции:

```
tvInfo.setText(getResources().getString(R.string.ahead));
```

в данном случае в информационном *поле* появится *значение* строкового ресурса с именем `ahead`.

Осталось реализовать логику приложения в методе `onClick()`. Предлагаем написать код этого метода самостоятельно, для контроля в приложении предложен листинг, который содержит один из вариантов кода описанного приложения.

4.5 Немного о работе с эмулятором

При выполнении данной работы мы использовали эмулятор для проверки работоспособности приложения. Рассмотрим некоторые возможности, облегчающие и ускоряющие взаимодействие с виртуальным устройством.

Во-первых, существует набор полезных комбинаций клавиш для управления виртуальным устройством:

- **Alt+Enter** - разворачивает эмулятор до размеров экрана;
- **Ctrl+F11** - меняет ориентацию эмулятора с портретной на альбомную и обратно;
- **F8** - включает/выключает сеть.

Полный *список* комбинаций клавиш для работы с эмулятором можно найти по ссылке: http://developer.android.com/tools/help/emulator.html.

Во-вторых, кто бы ни работал с эмулятором, тот на себе прочувствовал насколько терпеливым надо быть, чтобы взаимодействовать с ним, так медленно он работает. Существует решение для ускорения работы эмулятора *Android* и этим решением является Intel *Hardware* Accelerated*Execution Manager* (Intel® HAXM).

Intel Hardware Accelerated Execution Manager (Intel® HAXM) - это *приложение* с поддержкой аппаратной виртуализации (гипервизор), которое использует технологию виртуализации Intel для ускорения эмуляции приложений *Android* на компьютере для разработки. (http://software.intel.com/ru-ru/android/articles/intel-hardware-accelerated-execution-manager)

Intel HAXM способен ускорить работу эмулятора для *x86* устройств. При этом эмулятор будет работать со скоростью, приближенной к скорости работы реального устройства, что поможет сократить время на *запуск* и отладку приложения. Подробно познакомиться с установкой Intel HAXM и настройкой эмулятора на работу с ускорителем можно в статье по ссылке: http://habrahabr.ru/company/intel/blog/146114/.

В-третьих, не стоит прерывать процесс запуска виртуального устройства, наберитесь терпения, на старых компьютерах первый *запуск AVD*может занять до 10 минут, на современных - от одной до трех минут. После того, как создали и запустили *виртуальное* устройство имеет смысл оставить его открытым, при всех последующих запусках приложения будет использоваться уже открытое *виртуальное* устройство, что позволит сэкономить время.

4.6 Заключение

В лабораторной работе рассмотрен процесс разработки простого приложения переднего плана. Описано создание активности, настройка интерфейса и реализация логики приложения. Других компонентов в приложении не предусмотрено. В последующих работах будут рассматриваться приложения, содержащие несколько активностей. Смешанные приложения, работающие на переднем плане и при этом поддерживающие сервисы, работающие в фоновом режиме.

ЗАДАНИЕ на лабораторную работу № 3

Составьте приложение, в котором имеется:

1. ActionBar,
2. кнопка HOME,
3. меню с не менее чем семью items, с двумя группами кнопок и тремя исполнительными иконками на ActionBar. В группах кнопок, при установке флажка на одном item, на других флажки должны сниматься.
4. Каждая item должна выводить всплывающее или выпадающее меню второго уровня, а исполнительная иконка, должны выводить, при нажатии, сообщение.
5. Кроме того, в приложении должно быть одно контекстное меню, не менее чем из трех пунктов, на которые тоже выводится сообщение

Приложение 1.

```
package com.example.projectn;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.*;

public class MainActivity extends Activity {
  TextView tvInfo;
  EditText etInput;
  Button bControl;

  int guess;
  boolean gameFinished;

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

    tvInfo = (TextView)findViewById(R.id.textView1);
    etInput = (EditText)findViewById(R.id.editText1);
    bControl = (Button)findViewById(R.id.button1);

    guess = (int)(Math.random()*100);
    gameFinished = false;
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
  }
  public void onClick(View v){
    if (!gameFinished){
      int inp=Integer.parseInt(etInput.getText().toString());
      if (inp > guess)
        tvInfo.setText(getResources().getString(R.string.ahead));
      if (inp < guess)
        tvInfo.setText(getResources().getString(R.string.behind));
      if (inp == guess)
      {
        tvInfo.setText(getResources().getString(R.string.hit));
```

```
            bControl.setText(getResources().getString(R.string.play_more));
            gameFinished = true;
          }
        }
        else
        {
          guess = (int)(Math.random()*100);
          bControl.setText(getResources().getString(R.string.input_value));
          tvInfo.setText(getResources().getString(R.string.try_to_guess));
          gameFinished = false;
        }
        etInput.setText("");
      }
    }
```