# ПЕРВОЕ ANDROID-ПРИЛОЖЕНИЕ

В этом разделе будет построено приложение, которое называется DriodQuest. Оно представляет собой викторину на тему: «Хорошо ли пользователь знает Android». Пользователь отвечает на вопрос, нажимая кнопку «Да» или «Нет», а DriodQuest мгновенно сообщает ему результат.

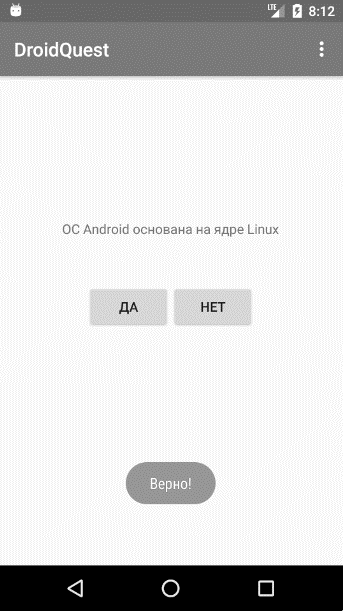
На рисунке 2.1 показан результат нажатия кнопки «Да».

Рисунок 2.1 – Приложение DriodQuest

**Основы построения приложения**

Приложение DriodQuest состоит из активности (activity) и макета

(layout):

Активность представлена экземпляром Activity — класса из Android SDK. Она отвечает за взаимодействие пользователя с информацией на экране.

Чтобы реализовать функциональность, необходимую приложению, разработчик пишет субклассы Activity. В простом приложении бывает достаточно одного субкласса; в сложном приложении их может потребоваться несколько.

DriodQuest — простое приложение, поэтому в нем используется всего один субкласс Activity с именем QuestActivity. Класс QuestActivity управляет пользовательским интерфейсом, изображенным на рисунке 2.1.

Макет определяет набор объектов пользовательского интерфейса и их расположение на экране. Приложение DriodQuest включает файл макета с именем activity\_quest.xml. Разметка XML в этом файле определяет пользовательский интерфейс, изображенный на рисунке 2.1.

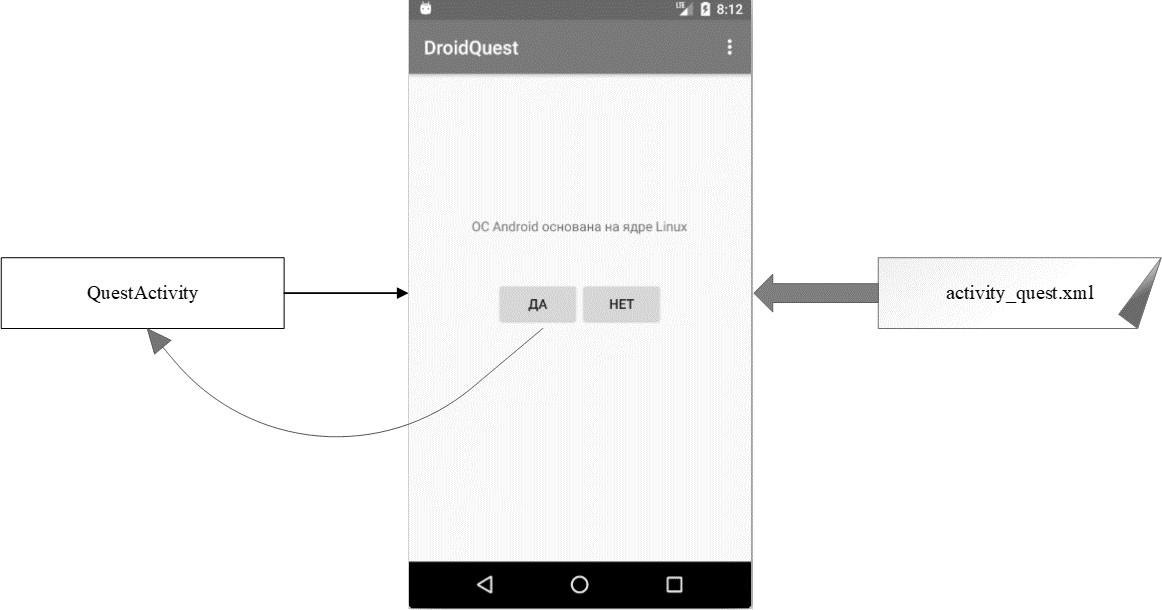
Отношения между QuestActivity и activity\_quest.xml изображены на рисунке 2.2.

Рисунок 2.2 – QuestActivity управляет интерфейсом, определяемым в файле activity\_quest.xml

**Создание проекта Android**

Проект Android содержит файлы, из которых состоит приложение. Чтобы создать новый проект, надо открыть Android Studio. Если Android Studio запускается на компьютере впервые, то на экране появляется диалоговое окно c приветствием (рисунок 2.3).

Выберитается в диалоговом окне команда Start a new Android Studio project. Если диалоговое окно не отображается при запуске, значит, ранее уже создавались другие проекты. В таком случае выполняется команду FileNew Project....

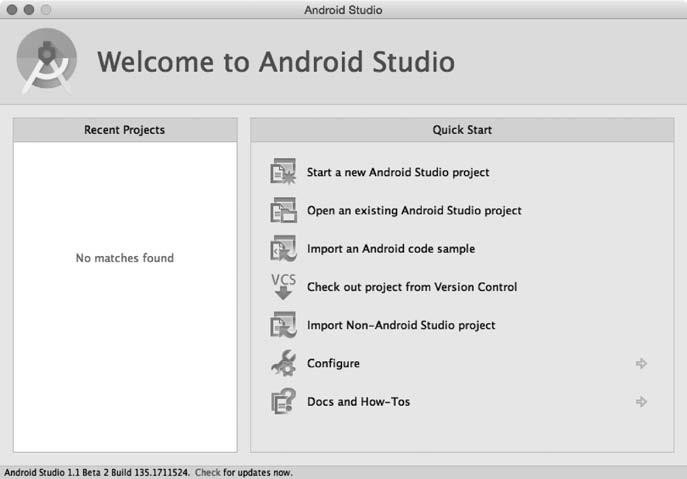


Рисунок 2.3 – Добро пожаловать в Android Studio

Открывается мастер создания проекта. На первом экране мастера вводится имя приложения **DriodQuest** (рисунок 2.4). В поле Company Domain введите строку ***android.rsue.ru*** (*android* – фамилия студента на латинице); сгенерированное имя пакета (Package Name), при этом автоматически меняется на *ru.rsue.android.driodquest*. В поле Project location вводится любая папка файловой системы.

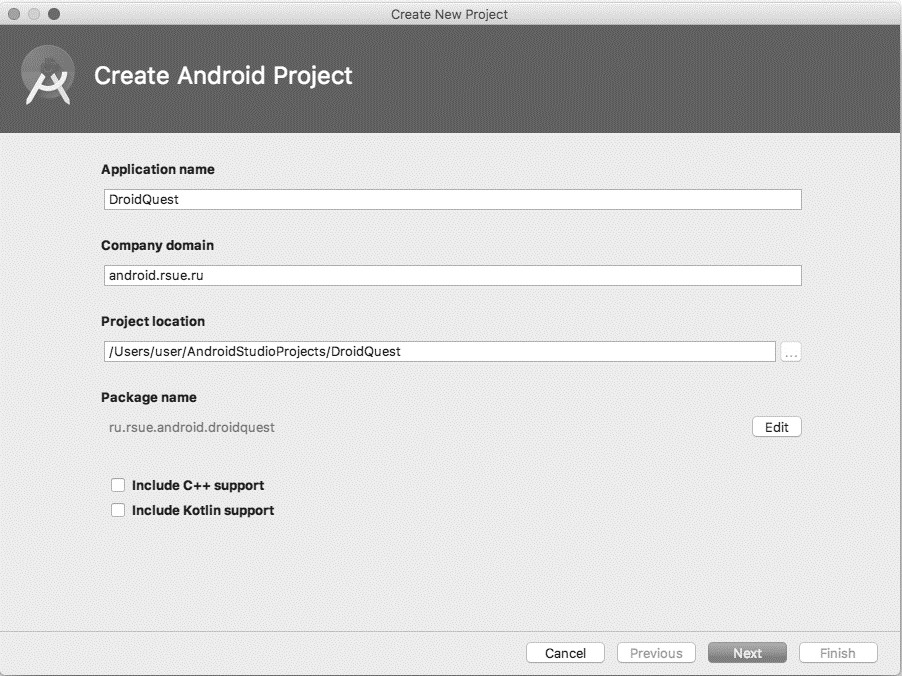


Рисунок 2.4 – Создание нового проекта

В имени пакета используется схема «обратного DNS», согласно которой доменное имя организации записывается в обратном порядке с присоединением суффиксов дополнительных идентификаторов. Эта схема обеспечивает уникальность имен пакетов и позволяет различать приложения на устройстве и в Google Play.

Щёлкнуть на кнопке Next. На следующем экране можно ввести дополнительную информацию об устройствах, которые необходимо поддерживать. Приложение DriodQuest будет поддерживать только телефоны, поэтому устанавливается только флажок Phone and Tablet. Выбирается в списке минимальная версия SDK API 21: Android 5.0 (Lollipop) (рисунок 2.5).

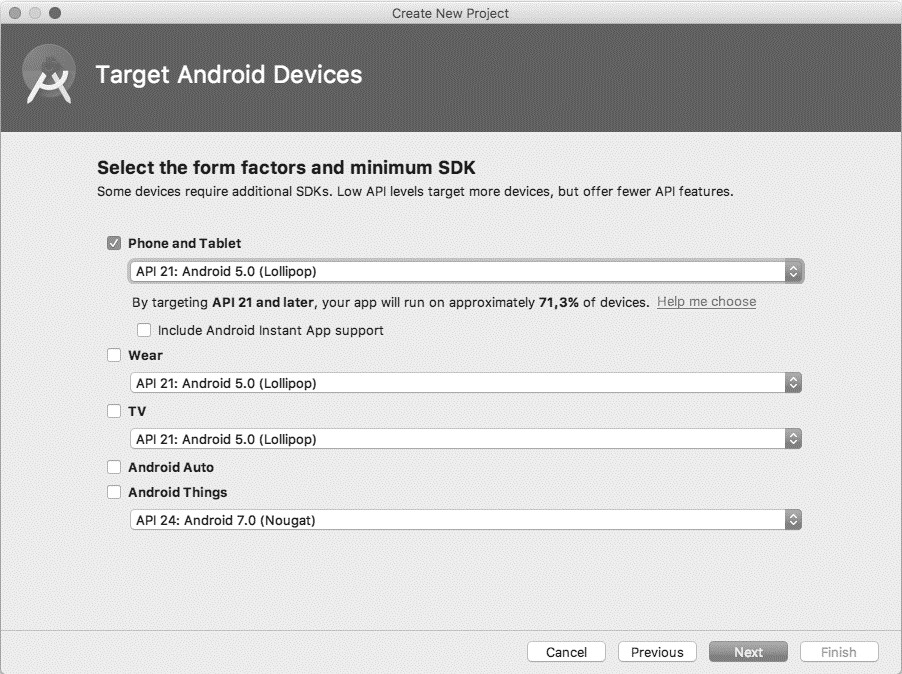


Рисунок 2.5 – Определение поддерживаемых устройств Щелкнуть на кнопке Next.

На следующем экране необходимо выбрать шаблон первого экрана DriodQuest (рисунок 2.6). Выбрать пустую активность (Blank Activity / Empty Activity) и щелкнуть на кнопке Next.

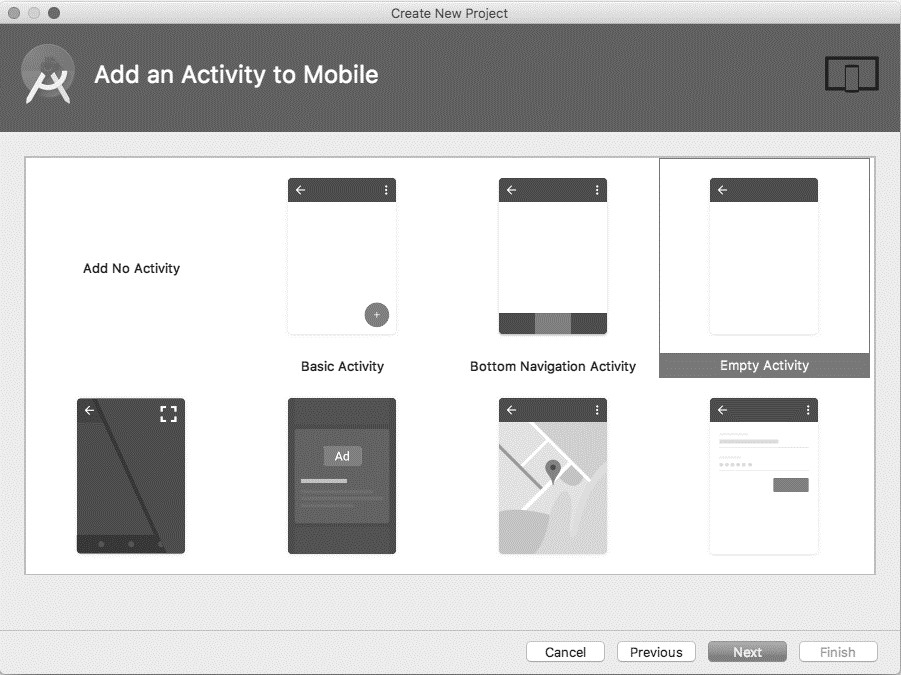


Рисунок 2.6 – Выбор типа активности

В последнем диалоговом окне мастера вводится имя субкласса активности QuestActivity (рисунок 2.7). Суффикс Activity в имени класса. Его присутствие не обязательно, но это очень полезное соглашение, которое стоит соблюдать.

Имя макета автоматически заменяется на activity\_quest в соответствии с переименованием активности. Имя макета записывается в порядке, обратном имени активности; в нем используются символы нижнего регистра, а слова разделяются символами подчеркивания.

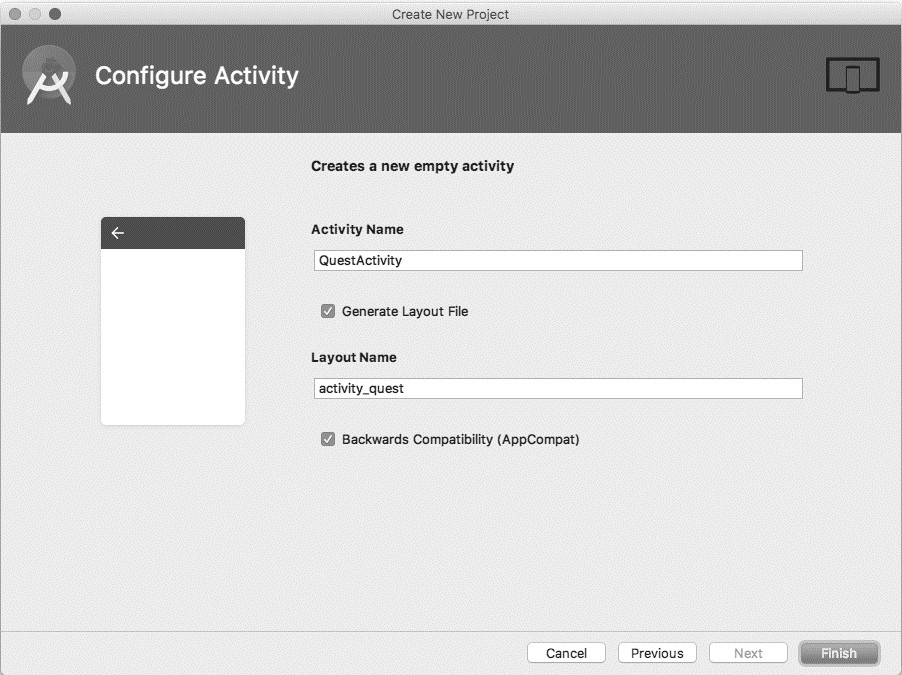


Рисунок 2.7 – Настройка новой активности

Щёлкнуть на кнопке Finish. Среда Android Studio создает и открывает новый проект.

**Навигация в Android Studio**

Среда Android Studio открывает созданный проект в окне, изображенном на рисунке 2.8.

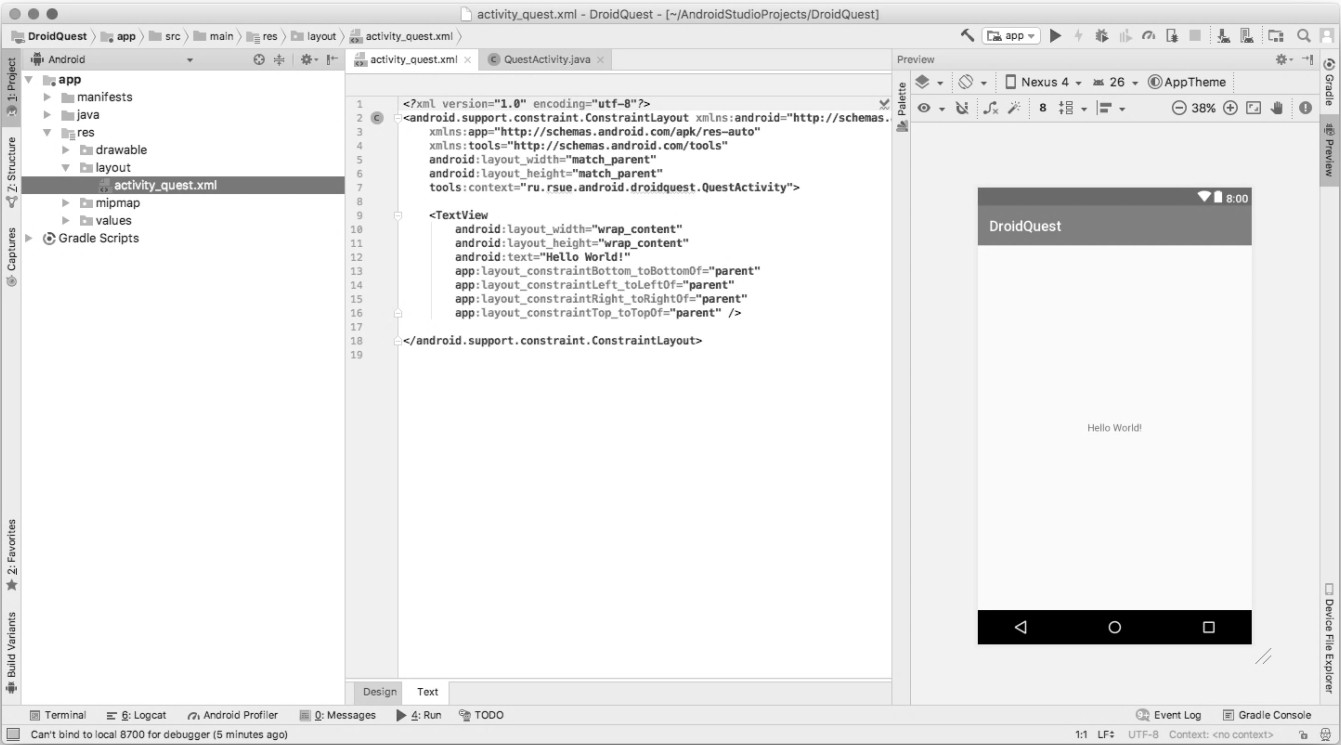


Рисунок 2.8 – Окно нового проекта

Различные части окна проекта называются панелями.

Слева располагается окно инструментов Project. В нем выполняются операции с файлами, относящимися к проекту.

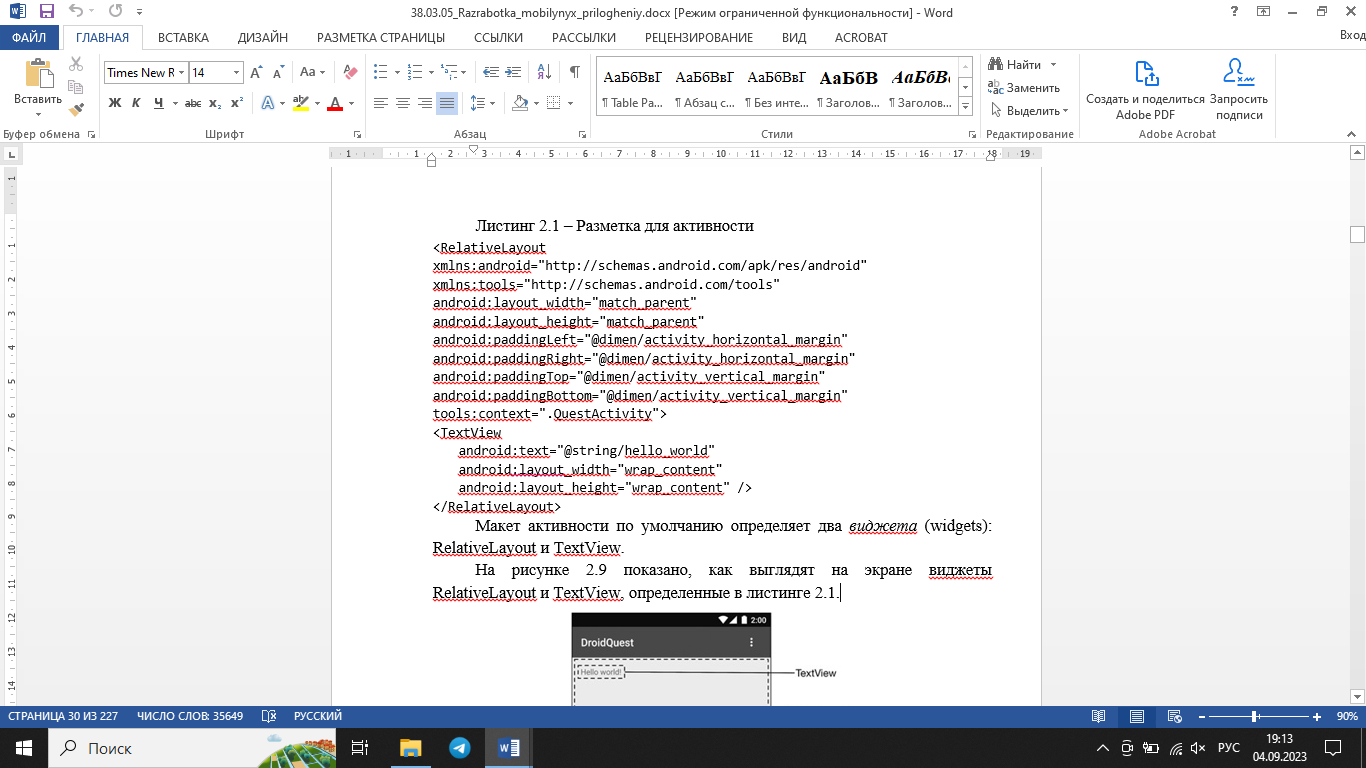
В середине находится панель редактора, в котором Android Studio открывает в файл activity\_quest.xml. (Если в редакторе выводится изображение, щёлкнуть на вкладке Text в нижней части панели.) На панели в правой части окна показано, как выглядит содержимое файла в режиме предварительного просмотра.

Видимостью различных панелей можно управлять, щелкая на их именах на полосе кнопок инструментов у левого, правого или нижнего края экрана. Также для многих панелей определены специальные комбинации клавиш. Если полосы с кнопками не отображаются, щёлкнуть на серой квадратной кнопке в левом нижнем углу главного окна или выполнить команду ViewTool Buttons.

Построение макета пользовательского интерфейса

На данный момент файл activity\_quest.xml определяет разметку для активности по умолчанию. Разметка шаблона часто изменяется, но XML будет выглядеть примерно так, как показано в листинге 2.1.

Листинг 2.1 – Разметка для активности



Макет активности по умолчанию определяет два виджета (widgets): RelativeLayout и TextView.

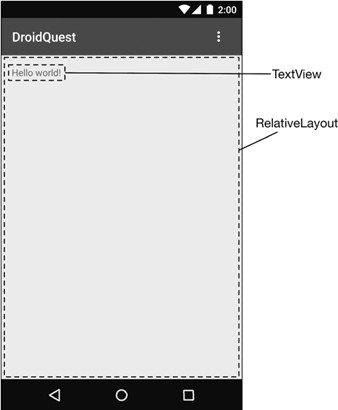
На рисунке 2.9 показано, как выглядят на экране виджеты RelativeLayout и TextView, определенные в листинге 2.1.

Рисунок 2.9 – Виджеты по умолчанию на экране

В интерфейсе QuestActivity будут задействованы пять виджетов: вертикальный виджет LinearLayout; TextView; горизонтальный виджет LinearLayout; две кнопки Button.

На рисунке 2.10 показано, как из этих виджетов образуется интерфейс QuestActivity.

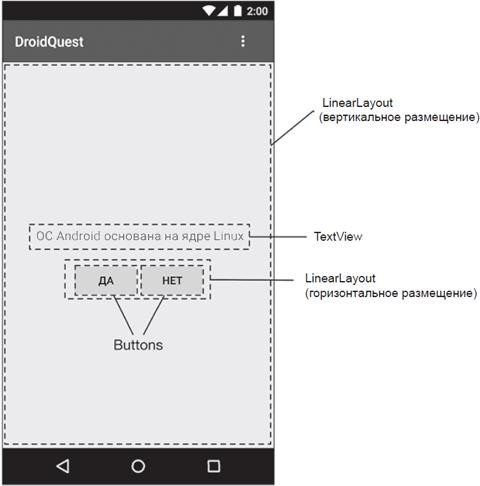
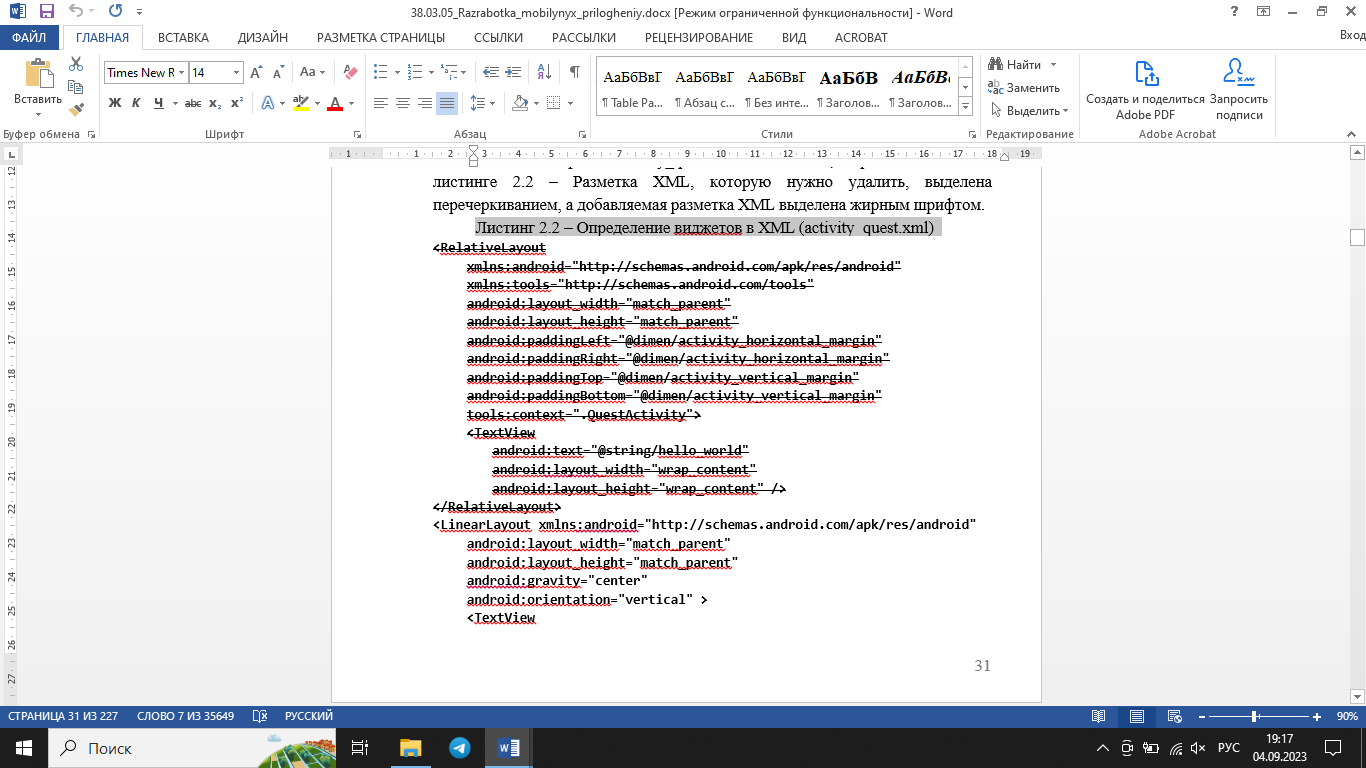


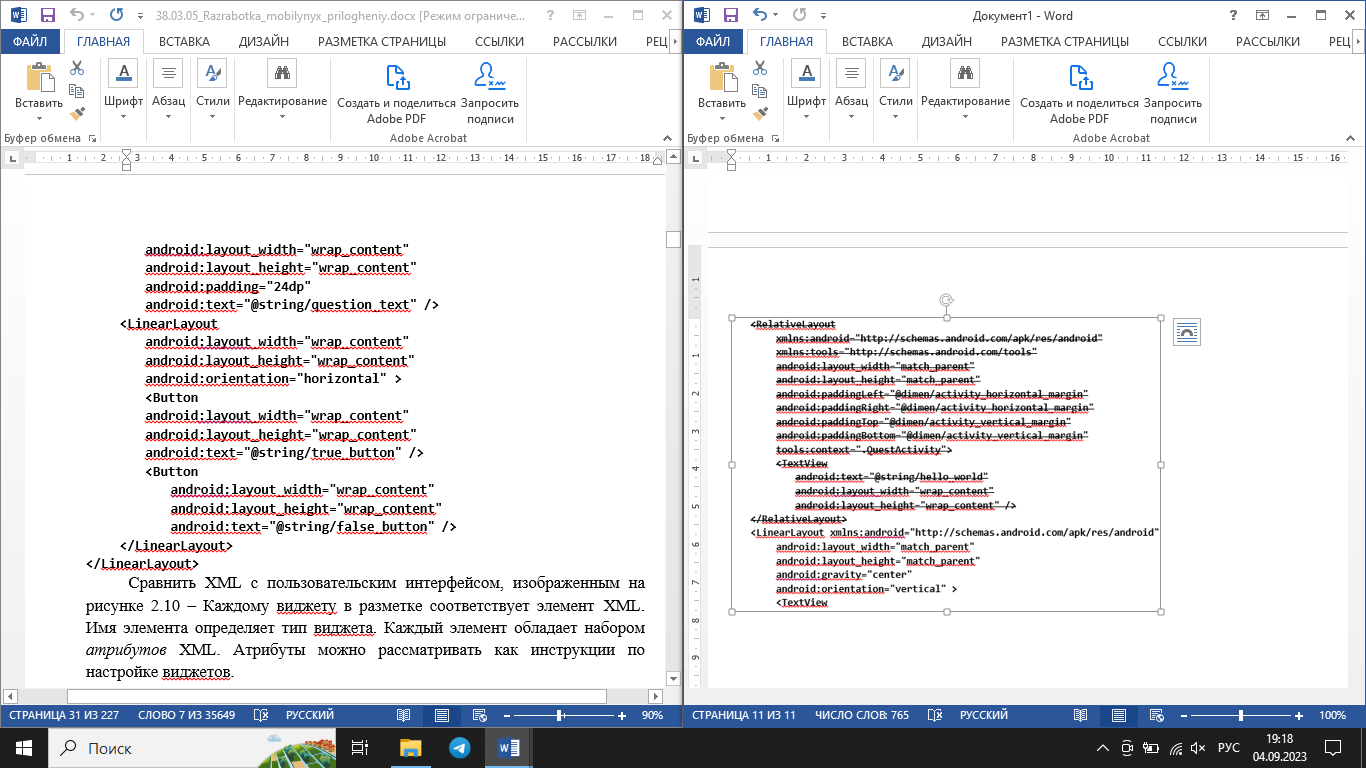
Рисунок 2.10 – Запланированное расположение виджетов на экране

Теперь нужно определить эти виджеты в файле activity\_quest.xml.

Внести в файл activity\_quest.xml изменения, представленные в листинге 2.2 – Разметка XML, которую нужно удалить, выделена перечеркиванием, а добавляемая разметка XML выделена жирным шрифтом.

Листинг 2.2 – Определение виджетов в XML (activity\_quest.xml)





Сравнить XML с пользовательским интерфейсом, изображенным на рисунке 2.10 – Каждому виджету в разметке соответствует элемент XML. Имя элемента определяет тип виджета. Каждый элемент обладает набором атрибутов XML. Атрибуты можно рассматривать как инструкции по настройке виджетов.

**Иерархия представлений**

Виджеты входят в иерархию объектов View, называемую иерархией представлений. На рисунке 2.11 изображена иерархия виджетов для разметки XML из листинга 2.2.

Корневым элементом иерархии представлений в этом макете является элемент LinearLayout. В нем должно быть указано пространство имен XML ресурсов Android http://schemas.android.com/apk/res/android.

LinearLayout наследует от субкласса View с именем ViewGroup. Виджет ViewGroup предназначен для хранения и размещения других виджетов. LinearLayout используется в тех случаях, когда необходимо выстроить виджеты в один столбец или строку. Другие субклассы ViewGroup

— FrameLayout, TableLayout и RelativeLayout.

Если виджет содержится в ViewGroup, он называется потомком (child) ViewGroup. Корневой элемент LinearLayout имеет двух потомков: TextView и другой элемент LinearLayout. У LinearLayout имеются два собственных потомка Button.

**Атрибуты виджетов**

Рассмотрим некоторые атрибуты, используемые для настройки виджетов.

android:layout\_width и android:layout\_height

Атрибуты android:layout\_width и android:layout\_height, определяющие ширину и высоту, необходимы практически для всех разновидностей виджетов. Как правило, им задаются значения match\_parent или wrap\_content:

 match\_parent — размеры представления определяются размерами родителя;

 wrap\_content — размеры представления определяются размерами содержимого.

(Иногда в разметке встречается значение fill\_parent. Это устаревшее значение эквивалентно match\_parent.)

В корневом элементе LinearLayout атрибуты ширины и высоты равны match\_parent. Элемент LinearLayout является корневым, но у него все равно есть родитель — представление, которое предоставляет Android для размещения иерархии представлений данного приложения.

У других виджетов макета ширине и высоте задается значение wrap\_content. На рисунке 2.10 показано, как в этом случае определяются их размеры.

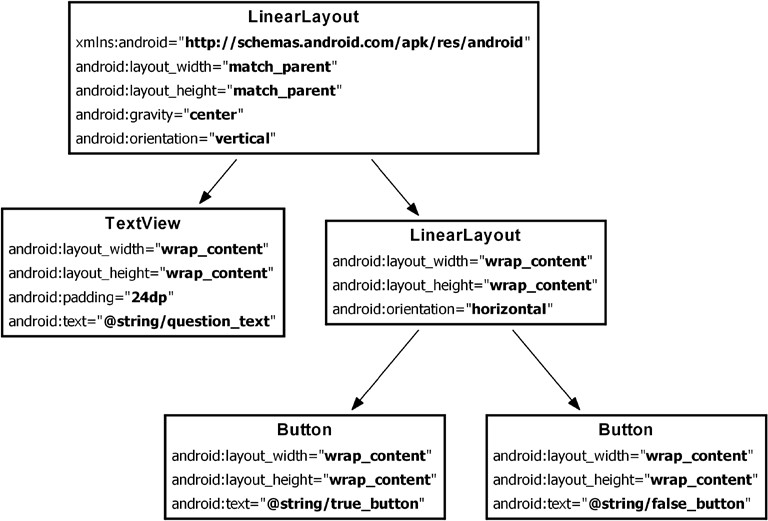


Рисунок 2.11 – Иерархия виджетов и атрибутов

Виджет TextView содержит чуть больше текста из-за атрибута android:padding="24dp". Этот атрибут приказывает виджету добавить заданный отступ вокруг содержимого при определении размера, чтобы текст вопроса не соприкасался с кнопкой (dp — это пикселы, не зависящие от плотности (density-independent pixels)).

android:orientation

Атрибут android:orientation двух виджетов LinearLayout определяет, как будут выстраиваться потомки — по вертикали или горизонтали. Корневой элемент LinearLayout имеет вертикальную ориентацию; у его потомка LinearLayout горизонтальная ориентация.

Порядок определения потомков определяет порядок их отображения на экране. В вертикальном элементе LinearLayout потомок, определенный первым, располагается выше остальных. В горизонтальном элементе LinearLayout первый потомок является крайним левым. (Если только на устройстве не используется язык с письменностью справа налево, например, арабский или иврит; в этом случае первый потомок будет находиться в крайней правой позиции.)

android:text

Виджеты TextView и Button содержат атрибуты android:text. Этот атрибут сообщает виджету, какой текст должен в нем отображаться.

Значения атрибутов представляют собой не строковые литералы, а ссылки на строковые ресурсы.

Строковый ресурс — строка, находящаяся в отдельном файле XML, который называется строковым файлом. Виджету можно назначить фиксированную строку (например, android:text="True"), но так делать не стоит. Лучше размещать строки в отдельном файле, а затем ссылаться на них, так как использование строковых ресурсов упрощает локализацию.

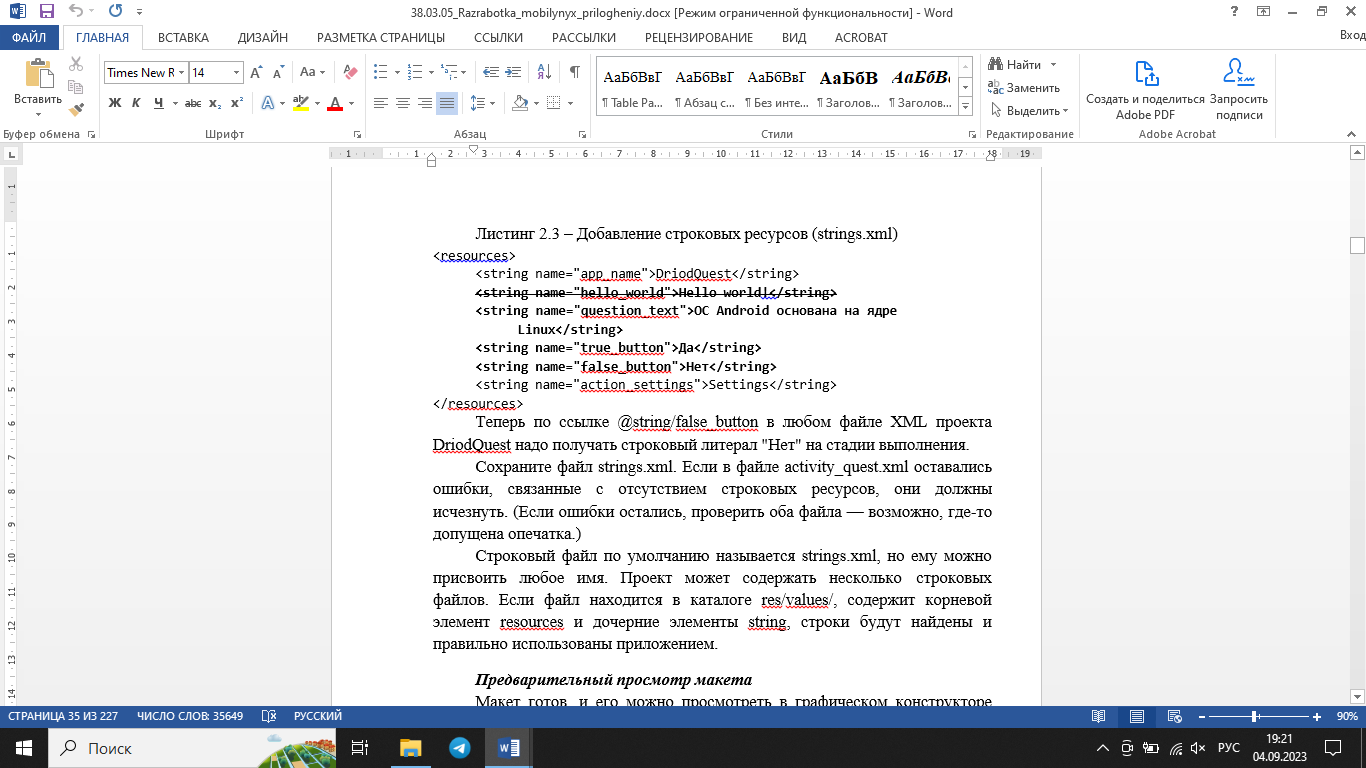
Строковые ресурсы, на которые ссылается activity\_quest.xml, еще не существуют. Это необходимо исправить.

**Создание строковых ресурсов**

Каждый проект включает строковый файл по умолчанию с именем strings.xml. Найти в окне Project каталог app/res/values, раскрыть его и открыть файл strings.xml.

В шаблон уже включено несколько строковых ресурсов. Удалить неиспользуемую строку с именем hello\_world и добавить три новые строки для макета.

Листинг 2.3 – Добавление строковых ресурсов (strings.xml)



Теперь по ссылке @string/false\_button в любом файле XML проекта DriodQuest надо получать строковый литерал "Нет" на стадии выполнения.

Сохраните файл strings.xml. Если в файле activity\_quest.xml оставались ошибки, связанные с отсутствием строковых ресурсов, они должны исчезнуть. (Если ошибки остались, проверить оба файла — возможно, где-то допущена опечатка.)

Строковый файл по умолчанию называется strings.xml, но ему можно присвоить любое имя. Проект может содержать несколько строковых файлов. Если файл находится в каталоге res/values/, содержит корневой элемент resources и дочерние элементы string, строки будут найдены и правильно использованы приложением.

**Предварительный просмотр макета**

Макет готов, и его можно просмотреть в графическом конструкторе (рисунок 2.12). Прежде всего надо убедиться в том, что файлы сохранены и не содержат ошибок. Затем вернуться к файлу activity\_quest.xml и открыть панель Preview при помощи вкладки в правой части редактора.

От разметки XML к объектам View

При создании проекта DriodQuest был автоматически создан субкласс Activity с именем QuestActivity. Файл класса QuestActivity находится в каталоге app/java (в котором хранится Java-код проекта).

В окне инструментов Project открыть каталог app/java, а затем содержимое пакета ru.rsue.android.driodquest. Открыть файл QuestActivity.java и просмотреть его содержимое (листинг 2.4).

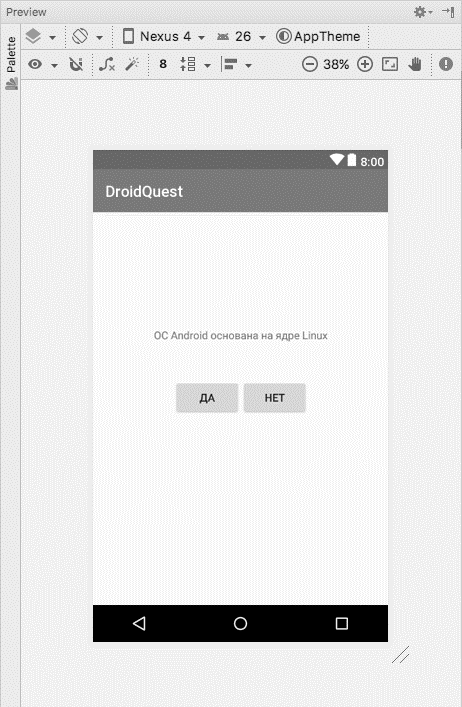
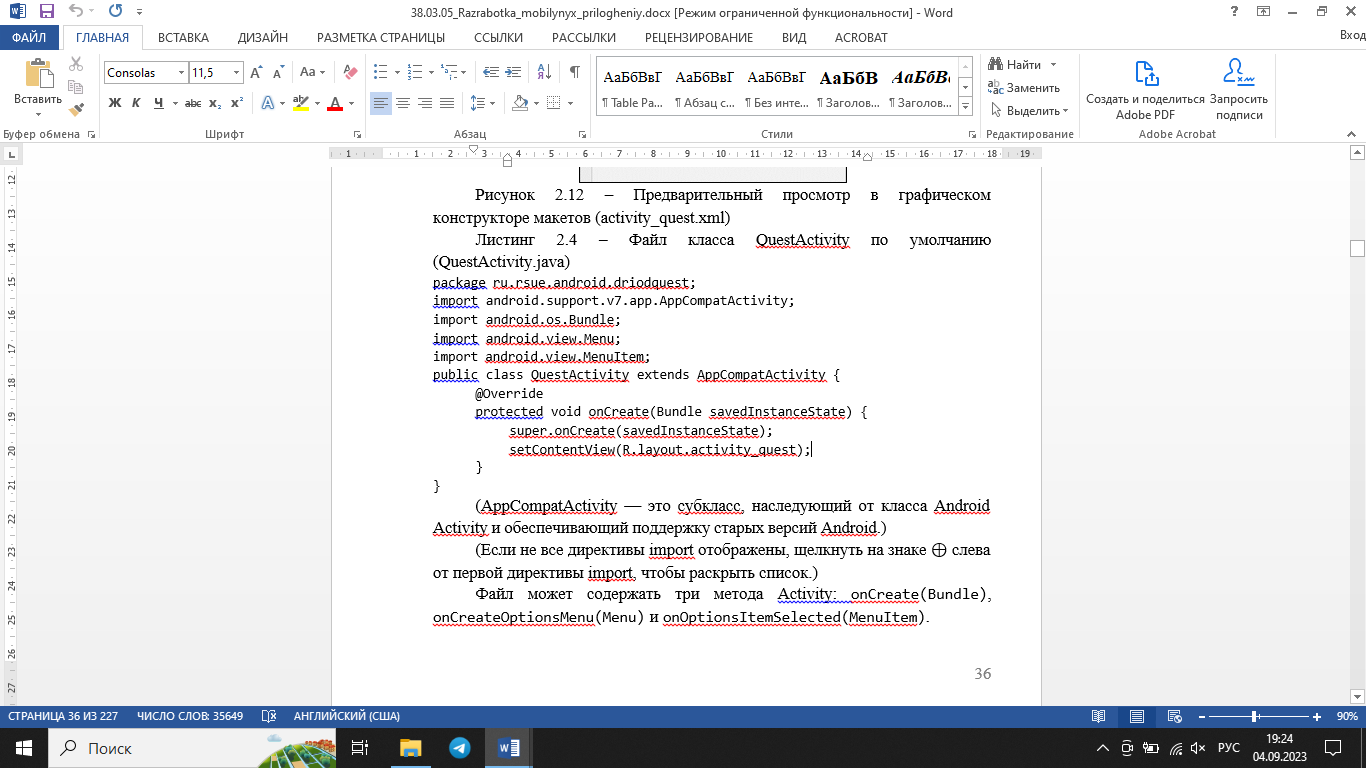


Рисунок 2.12 – Предварительный просмотр в графическом конструкторе макетов (activity\_quest.xml)

Листинг 2.4 – Файл класса QuestActivity по умолчанию (QuestActivity.java)



(AppCompatActivity — это субкласс, наследующий от класса Android

Activity и обеспечивающий поддержку старых версий Android.)

(Если не все директивы import отображены, щелкнуть на знаке ⊕ слева от первой директивы import, чтобы раскрыть список.)

Файл может содержать три метода Activity: onCreate(Bundle), onCreateOptionsMenu(Menu) и onOptionsItemSelected(MenuItem).

Метод onCreate(Bundle) вызывается при создании экземпляра субкласса активности. Такому классу нужен пользовательский интерфейс, которым он будет управлять. Чтобы предоставить классу активности его пользовательский интерфейс, следует вызвать метод Activity:

public void setContentView(int layoutResID);

Этот метод заполняет (inflates) макет и выводит его на экран. При заполнении макета создаются экземпляры всех виджетов в файле макета с параметрами, определяемыми его атрибутами. Чтобы указать, какой именно макет следует заполнить, необходимо передать идентификатор ресурса макета.

**Ресурсы и идентификаторы ресурсов**

Для обращения к ресурсу в коде используется его идентификатор ресурса. Макету приложения назначен идентификатор ресурса R.layout.activity\_quest.

Чтобы просмотреть текущие идентификаторы ресурсов проекта DriodQuest, необходимо сначала изменить режим представления проекта. По умолчанию Android Studio использует режим представления Android (рисунок 2.13). В этом режиме истинная структура каталогов проекта Android скрывается, чтобы можно было сосредоточиться на тех файлах и папках, которые чаще всего нужны программисту.

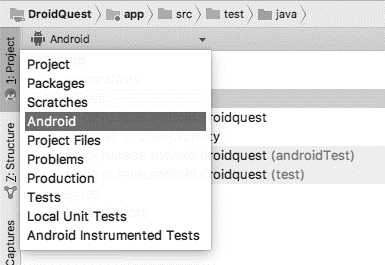
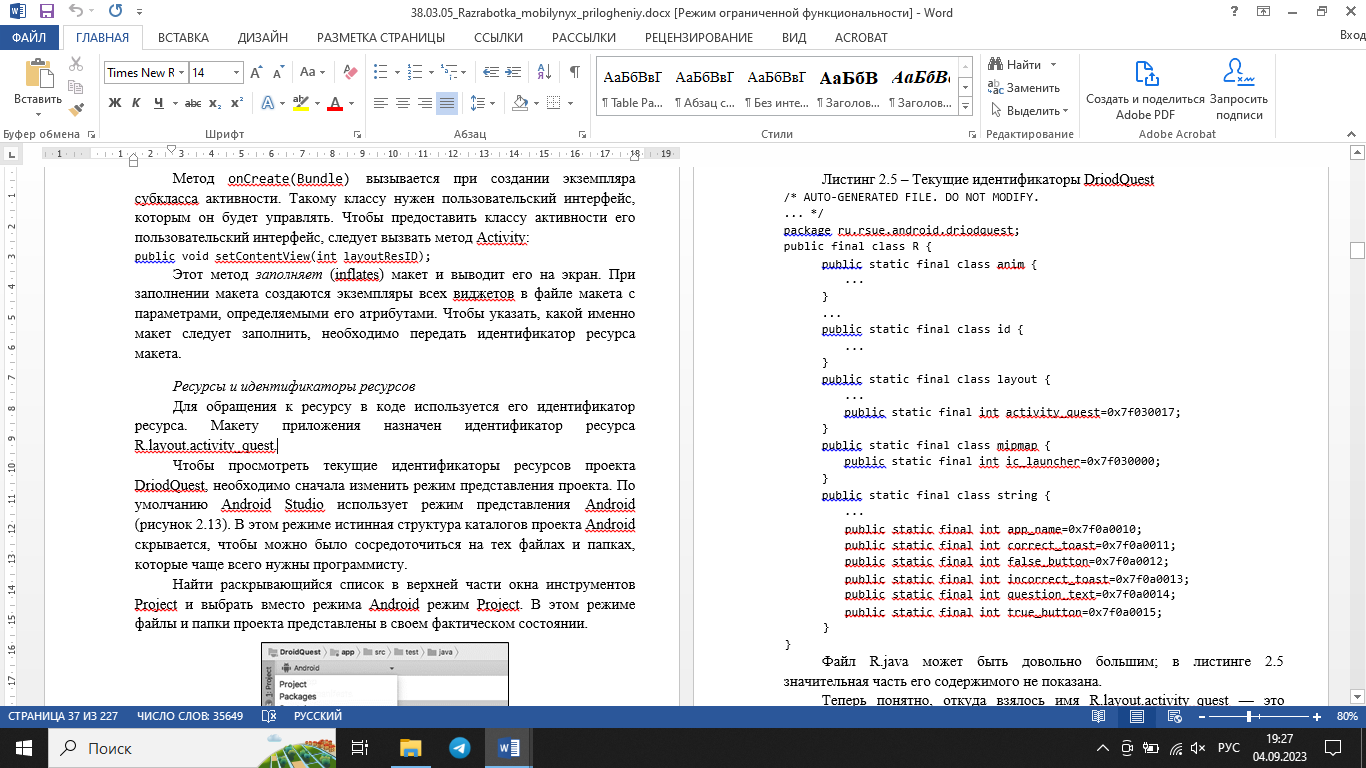
Найти раскрывающийся список в верхней части окна инструментов Project и выбрать вместо режима Android режим Project. В этом режиме файлы и папки проекта представлены в своем фактическом состоянии.

Рисунок 2.13 – Изменение режима представления проекта

Чтобы просмотреть ресурсы приложения DriodQuest, надо открыть содержимое каталога app/build/generated/source/r/debug. В этом каталоге найти имя пакета проекта и открыть файл R.java из этого пакета. Поскольку этот файл генерируется процессом сборки Android, не следует его изменять, о чем деликатно предупреждает надпись в начале файла.

Листинг 2.5 – Текущие идентификаторы DriodQuest



Файл R.java может быть довольно большим; в листинге 2.5 значительная часть его содержимого не показана.

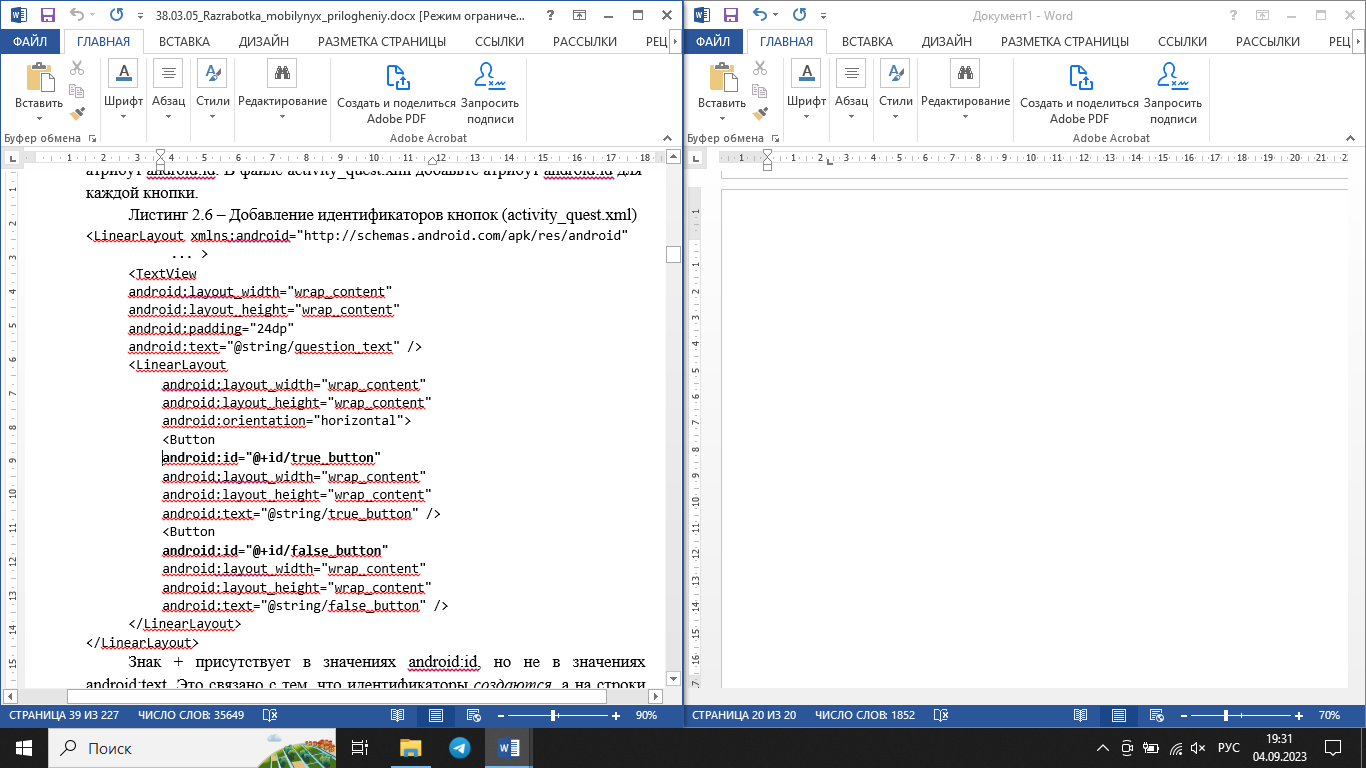
Теперь понятно, откуда взялось имя R.layout.activity\_quest — это целочисленная константа с именем activity\_quest из внутреннего класса layout класса R.

Строкам также назначаются идентификаторы ресурсов. В приложении еще не было ссылок на строки в коде, но эти ссылки обычно выглядят так: setTitle(R.string.app\_name);

Android генерирует идентификатор ресурса для всего макета и для каждой строки, но не для отдельных виджетов из файла activity\_quest.xml. Не каждому виджету нужен идентификатор ресурса.

Прежде чем генерировать идентификаторы ресурсов, переключитесь обратно в режим представления Android. Чтобы сгенерировать идентификатор ресурса для виджета, включите в определение виджета атрибут android:id. В файле activity\_quest.xml добавьте атрибут android:id для каждой кнопки.

Листинг 2.6 – Добавление идентификаторов кнопок (activity\_quest.xml)



Знак + присутствует в значениях android:id, но не в значениях android:text. Это связано с тем, что идентификаторы создаются, а на строки только ссылаемся.

**Подключение виджетов к программе**

Теперь, когда кнопкам назначены идентификаторы ресурсов, к ним можно обращаться в QuestActivity. Все начинается с добавления двух переменных.

Введите следующий код в QuestActivity.java. (Не используйте автозавершение; введите его самостоятельно.) После сохранения файла выводятся два сообщения об ошибках.

Листинг 2.7 – Добавление полей (QuestActivity.java)

public class QuestActivity extends AppCompatActivity {

private Button mTrueButton;

private Button mFalseButton;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_quest);

}

}

Обратить внимание на префикс m у имен двух полей (переменных экземпляров). Этот префикс соответствует схеме формирования имен Android.

Навести указатель мыши на красные индикаторы ошибок. Они сообщают об одинаковой проблеме: «Не удается разрешить символическое имя Button» (Cannot resolve symbol ‘Button’).

Чтобы избавиться от ошибок, следует импортировать класс android.widget.Button в QuestActivity.java. Ввести следующую директиву импортирования в начале файла:

import android.widget.Button;

А можно пойти по простому пути и поручить эту работу Android Studio. Нажать Alt+Enter — волшебство IntelliJ приходит на помощь. Новая директива import теперь появляется под другими директивами в начале файла. Этот прием часто бывает полезным, если код работает не так, как положено.

Теперь можно подключить виджеты-кнопки. Процедура состоит из двух шагов:

 получение ссылок на заполненные объекты View;

 назначение для этих объектов слушателей, реагирующих на действия пользователя.

**Получение ссылок на виджеты**

В классе активности можно получить ссылку на заполненный виджет, для чего используется следующий метод Activity:

public View findViewById(int id);

Метод получает идентификатор ресурса виджета и возвращает объект View.

В файле QuestActivity.java по идентификаторам ресурсов кнопок можно получить заполненные объекты и присвоить их полям. Возвращенный объект View перед присваиванием необходимо преобразовать в Button.

Листинг 2.8 – Получение ссылок на виджеты (QuestActivity.java)

public class QuestActivity extends AppCompatActivity {

...

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_quest);

mTrueButton = (Button) findViewById(R.id.true\_button);

mFalseButton = (Button) findViewById(R.id.false\_button);

}

...

}

**Назначение слушателей**

Приложения Android обычно управляются событиями (event-driven). В отличие от программ командной строки или сценариев, такие приложения запускаются и ожидают наступления некоторого события, например, нажатия кнопки пользователем. (События также могут инициироваться ОС или другим приложением, но события, инициируемые пользователем, наиболее очевидны.)

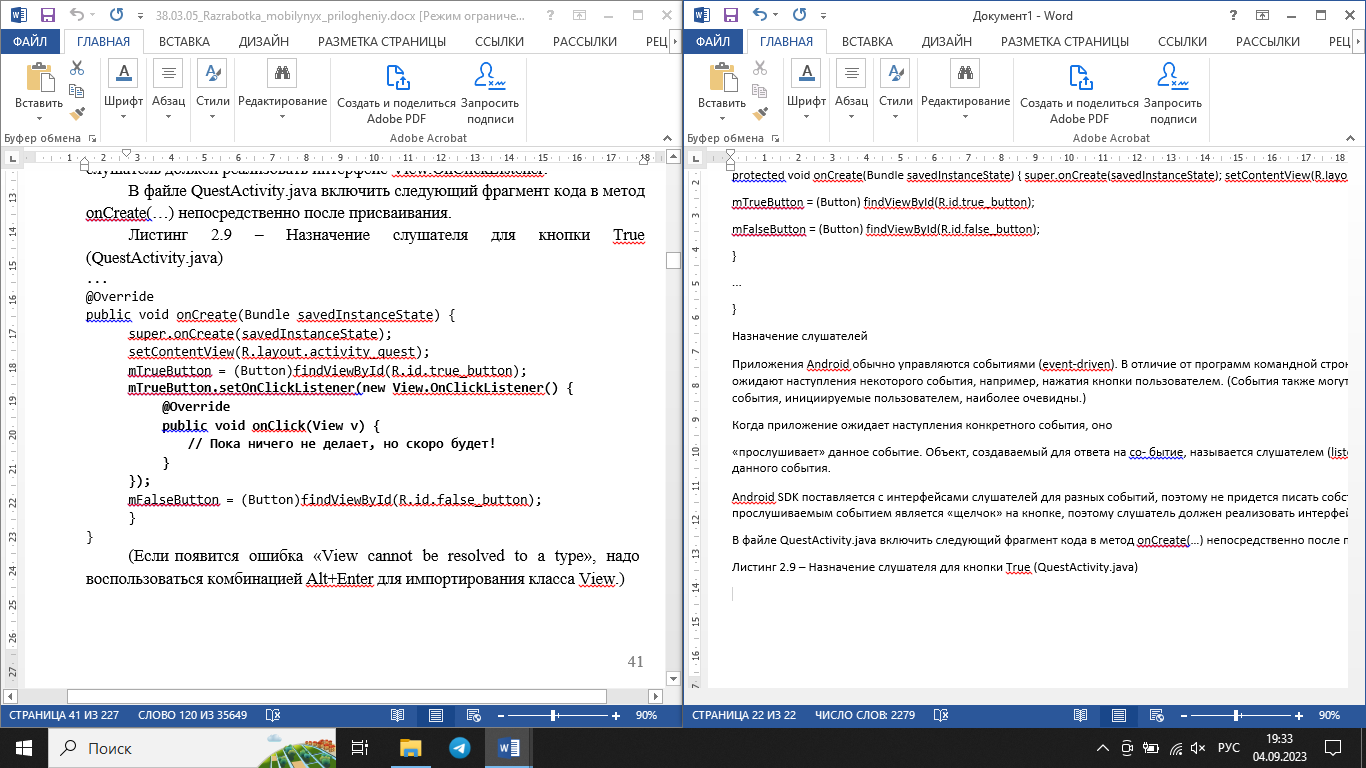
Когда приложение ожидает наступления конкретного события, оно

«прослушивает» данное событие. Объект, создаваемый для ответа на со- бытие, называется слушателем (listener). Такой объект реализует интерфейс слушателя данного события.

Android SDK поставляется с интерфейсами слушателей для разных событий, поэтому не придется писать собственные реализации. В данном случае прослушиваемым событием является «щелчок» на кнопке, поэтому слушатель должен реализовать интерфейс View.OnClickListener.

В файле QuestActivity.java включить следующий фрагмент кода в метод onCreate(…) непосредственно после присваивания.

Листинг 2.9 – Назначение слушателя для кнопки True (QuestActivity.java)



(Если появится ошибка «View cannot be resolved to a type», надо воспользоваться комбинацией Alt+Enter для импортирования класса View.)

В листинге 2.9 назначается слушатель, информирующий о нажатии виджета Button с именем mTrueButton. Метод setOnClickListener(OnClickListener) получает в аргументе слушателя, а конкретнее — объект, реализующий OnClickListener.

**Анонимные внутренние классы**

Слушатель реализован в виде анонимного внутреннего класса. Возможно, синтаксис не очевиден; просто запомните: все, что заключено во внешнюю пару круглых скобок, передается setOnClickListener(OnClickListener). В круглых скобках создается новый безымянный класс, вся реализация которого передается вызываемому методу.

mTrueButton.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

// Пока ничего не делает, но скоро будет!

});

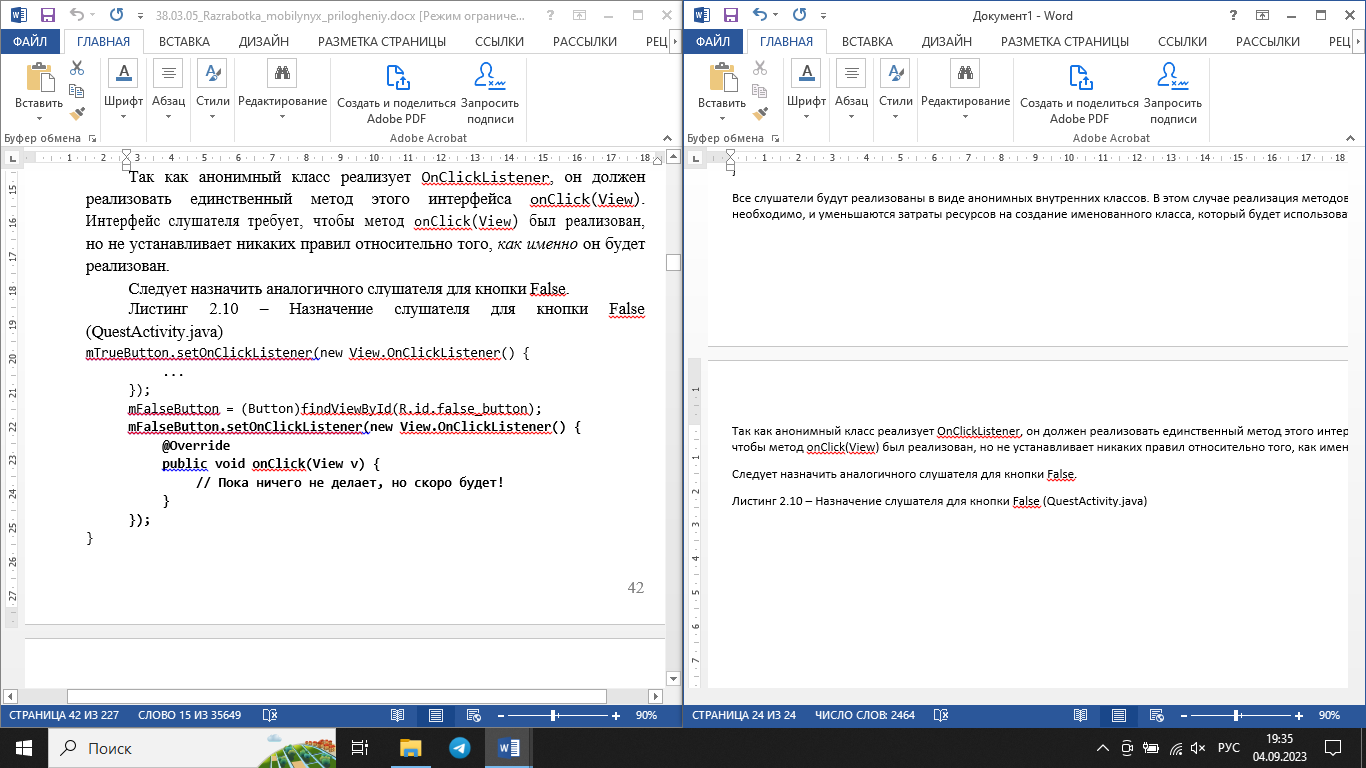
}

Все слушатели будут реализованы в виде анонимных внутренних классов. В этом случае реализация методов слушателя находится непосредственно там, где это необходимо, и уменьшаются затраты ресурсов на создание именованного класса, который будет использоваться только в одном месте.

Так как анонимный класс реализует OnClickListener, он должен реализовать единственный метод этого интерфейса onClick(View). Интерфейс слушателя требует, чтобы метод onClick(View) был реализован, но не устанавливает никаких правил относительно того, как именно он будет реализован.

Следует назначить аналогичного слушателя для кнопки False.

Листинг 2.10 – Назначение слушателя для кнопки False (QuestActivity.java)



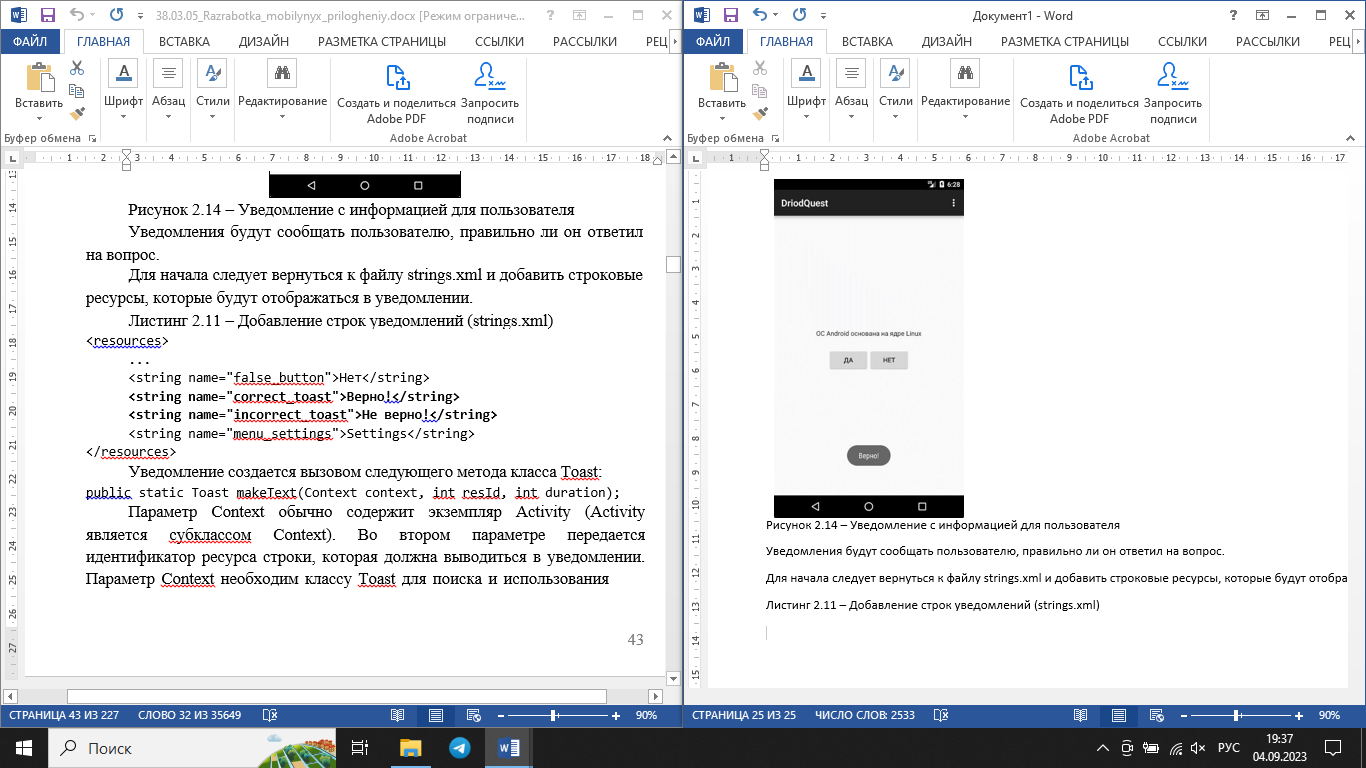
**Уведомления**

В данном приложении каждая кнопка будет выводить на экран временное уведомление (toast) — короткое сообщение, которое содержит какую-либо информацию для пользователя, но не требует ни ввода, ни действий (рисунок 2.14).

Рисунок 2.14 – Уведомление с информацией для пользователя 

Уведомления будут сообщать пользователю, правильно ли он ответил на вопрос. Для начала следует вернуться к файлу strings.xml и добавить строковые ресурсы, которые будут отображаться в уведомлении.

Листинг 2.11 – Добавление строк уведомлений (strings.xml)



Уведомление создается вызовом следующего метода класса Toast:

public static Toast makeText(Context context, int resId, int duration);

Параметр Context обычно содержит экземпляр Activity (Activity является субклассом Context). Во втором параметре передается идентификатор ресурса строки, которая должна выводиться в уведомлении. Параметр Context необходим классу Toast для поиска и использования идентификатора ресурса строки. Третий параметр обычно содержит одну из двух констант Toast, определяющих продолжительность пребывания уведомления на экране.

После того как объект уведомления будет создан, вызвать Toast.show(), чтобы уведомление появилось на экране.

В классе QuestActivity вызов makeText(…) будет присутствовать в слушателе каждой кнопки (листинг 2.12).

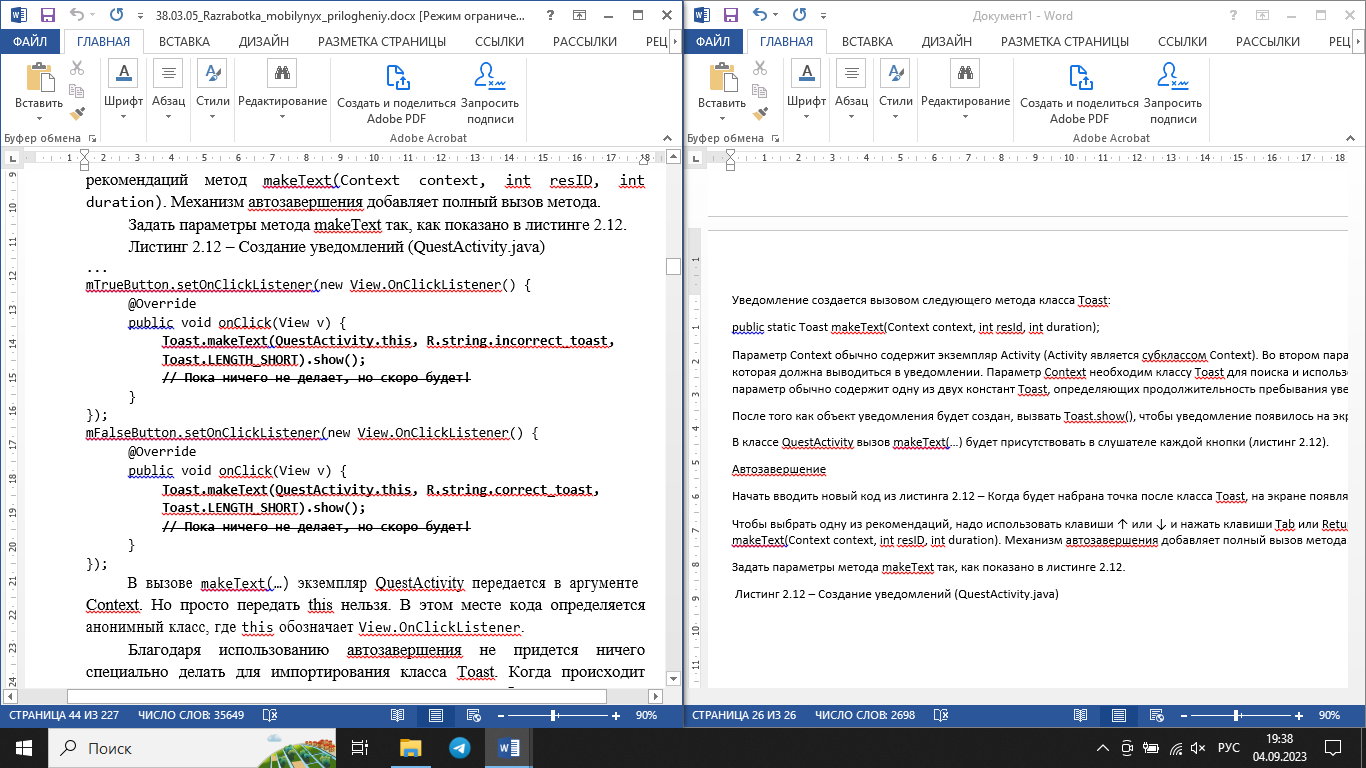
**Автозавершение**

Начать вводить новый код из листинга 2.12 – Когда будет набрана точка после класса Toast, на экране появляется список методов и констант класса Toast.

Чтобы выбрать одну из рекомендаций, надо использовать клавиши ↑ или ↓ и нажать клавиши Tab или Return/Enter. Выбрать в списке рекомендаций метод makeText(Context context, int resID, int duration). Механизм автозавершения добавляет полный вызов метода.

Задать параметры метода makeText так, как показано в листинге 2.12.

Листинг 2.12 – Создание уведомлений (QuestActivity.java)



В вызове makeText(…) экземпляр QuestActivity передается в аргументе Context. Но просто передать this нельзя. В этом месте кода определяется анонимный класс, где this обозначает View.OnClickListener.

Благодаря использованию автозавершения не придется ничего специально делать для импортирования класса Toast. Когда происходит соглашение на рекомендацию автозавершения, необходимые классы импортируются автоматически.

Сохраните внесенные изменения.

**Выполнение в эмуляторе**

Для запуска приложений Android необходимо устройство — физическое или виртуальное. Виртуальные устройства работают под управлением эмулятора Android, включенного в поставку средств разработчика.

Чтобы создать виртуальное устройство Android (AVD, Android Virtual Device), выполнить команду ToolsAndroidAVD Manager. Когда на экране появится окно AVD Manager, щелкнуть на кнопке Create Virtual Device... в левой части этого окна.

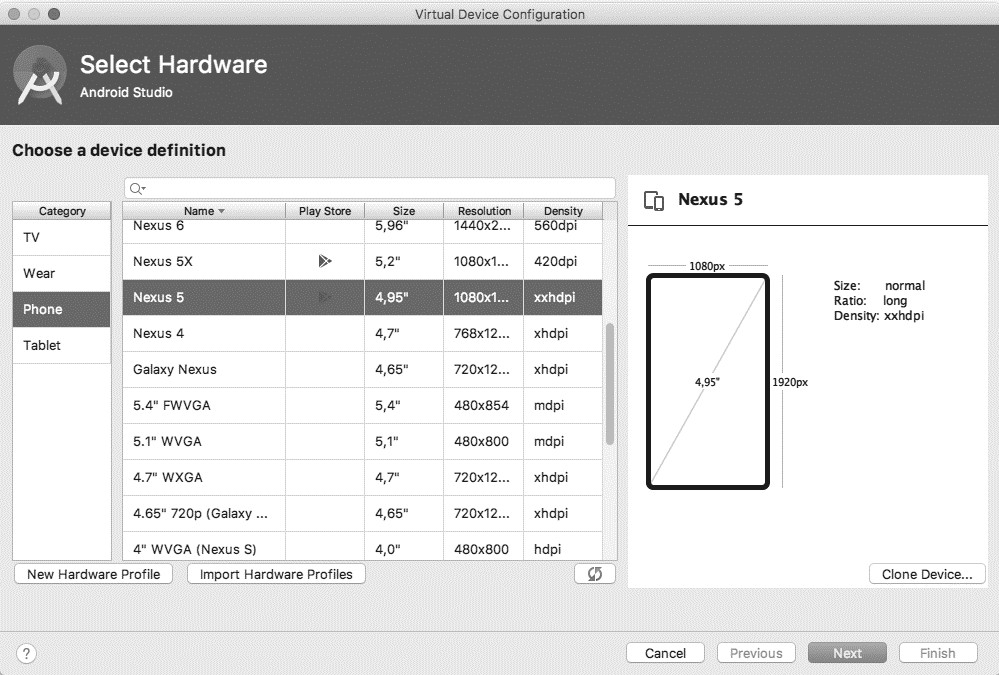
Открывается диалоговое окно с многочисленными параметрами настройки виртуального устройства. Выбрать эмуляцию устройства Nexus 5, как показано на рисунке 2.15. Щёлкнуть на кнопке Next.

Рисунок 2.15 – Выбор виртуального устройства

На следующем экране выбрать образ системы, на основе которого будет работать эмулятор. Выберите эмулятор x86 Marshmallow и щелкнуть на кнопке Next (рисунок 2.16).

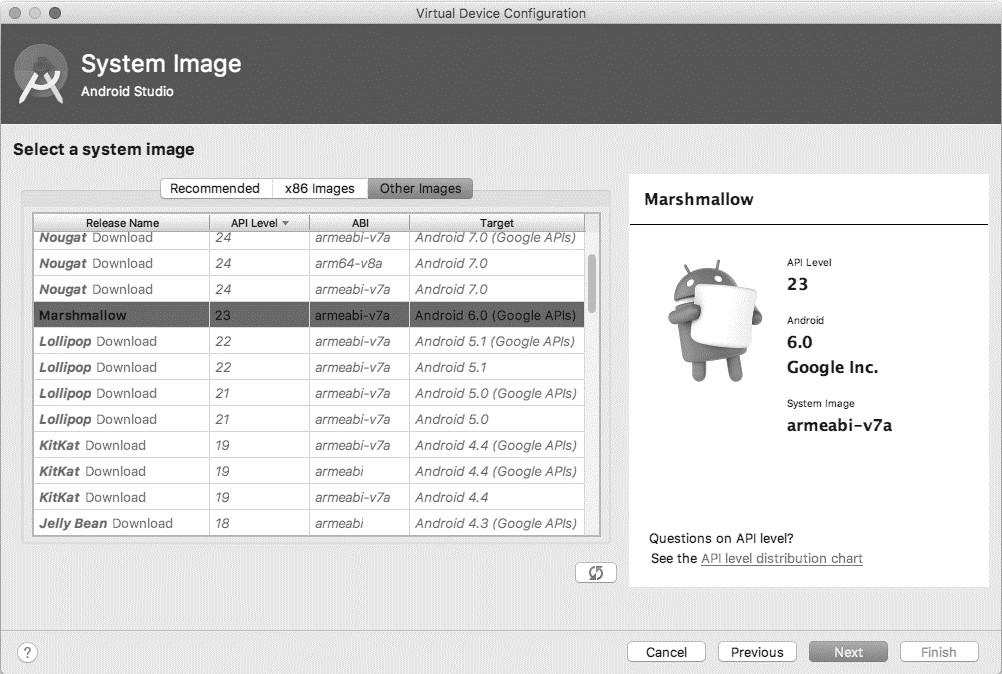


Рисунок 2.16 – Выбор образа системы

Наконец, просмотреть и при необходимости изменить свойства эмулятора (впрочем, свойства существующего эмулятора также можно изменить позднее). Пока необходимо присвоить эмулятору имя, и щёлкнуть на кнопке Finish (Рисунок 2.17).



Рисунок 2.17 – Настройка свойств эмулятора

Когда виртуальное устройство будет создано, в нем можно запустить приложение DriodQuest. На панели инструментов Android Studio щёлкнуть на кнопке Run (зеленый символ «воспроизведение») или нажать Ctrl+R. Android Studio находит созданное виртуальное устройство, запускает его, устанавливает на нем пакет приложения и запускает приложение.

Возможно, запуск эмулятора потребует некоторого времени, но вскоре приложение DriodQuest запустится на созданном виртуальном устройстве. Понажимать кнопки и оценить уведомления. (Если приложение запускается, когда вас нет поблизости, возможно, вам придется разблокировать AVD после возвращения. AVD работает как настоящее устройство и блокируется после некоторого времени бездействия.)

Если при запуске DriodQuest или нажатии кнопки происходит сбой, в нижней части представления LogCat панели Android DDMS выводится полезная информация. (Если представление LogCat не открылось автоматически при запуске DriodQuest, его можно открыть при помощи кнопки Android в нижней части окна Android Studio.) Ищите исключения в журнале; они будут выделяться красным цветом.

Сравнить свой код с кодом раздела и попытаться найти источник проблемы. Затем снова запустите приложение.

*Не закрывать эмулятор*; не стоит ждать, пока он загружается, при каждом запуске, а можно остановить приложение кнопкой Back (˅-образная стрелка), а затем снова запустить приложение из Android Studio, чтобы протестировать изменения.