ЛАБОРАТОРНАЯ РАБОТА №4

[Эргономика мобильных приложений: пособие / М. М. Меженная [и др.] Э74 – Минск : БГУИР, 2017. – 64с.]

Activity: работа с элементами экрана, обработка нажатий на кнопки

Цель: формирование у студентов знаний и навыков по работе с элементами экрана, обработке нажатий кнопок.

План занятия

1. Изучить теоретические сведения.

2. Выполнить практическое задание по лабораторной работе.

3. Оформить отчёт и ответить на контрольные вопросы.

Теоретические сведения

Мобильное приложение и мобильная платформа. Мобильное приложение представляет собой специально разработанное под конкретную мобильную платформу (***Android, iOS, Windows Phone***) программное обеспечение. Такое приложение также называют нативным мобильным приложением: его разрабатывают на языке высокого уровня и компилируют в нативный код операционной системы, дающий максимальную производительность и функциональность. Нативные мобильные приложения распространяются через магазины (***AppStore, Windows Store, Google Play***), существенно расширяют способы монетизации бизнеса по сравнению с веб приложениями, однако характеризуются высокой стоимостью и временем разработки.

Современные мобильные приложения создаются для линейки мобильных устройств: смартфонов, планшетов, электронных книг, цифровых проигрывателей, наручных часов, игровых приставок, нетбуков, смартбуков, очков, а также для телевизоров (тем самым выходя за рамки исключительно мобильных устройств). Работу таких устройств обеспечивает мобильная операционная система (мобильная платформа), сочетающая в себе функциональность операционной системы для персонального компьютера с функциями для мобильных и карманных устройств: сенсорный экран, сотовая связь, ***Bluetooth, Wi-Fi, GPS***-навигация, камера, видеокамера, распознавание речи, диктофон, музыкальный плеер, ***NFC*** и инфракрасное дистанционное управление.

Наиболее распространённые операционные системы (платформы) для мобильных устройств: ***Android*** (платформа с открытым исходным кодом на основе ядра ***Linux*** и собственной реализации виртуальной машины ***Java*** от ***Google***), ***iOS*** (операционная система компании ***Apple***, основанная на микроядре ***Mach*** и используемая в смартфонах ***iPhone***), ***Windows Phone*** (разработка компании ***Microsoft***). Архитектура и основные компоненты мобильной платформы ***Android***. В рамках дисциплины «Эргономика мобильных приложений» рассматриваются вопросы проектирования интерфейсов и разработки мобильных приложений под платформу ***Android***, получившую наибольшее распространение в мире. Архитектуру ***Android*** принято делить на уровни ядра, библиотек и среды выполнения, каркаса приложений, собственно приложений (рисунок 1.1).

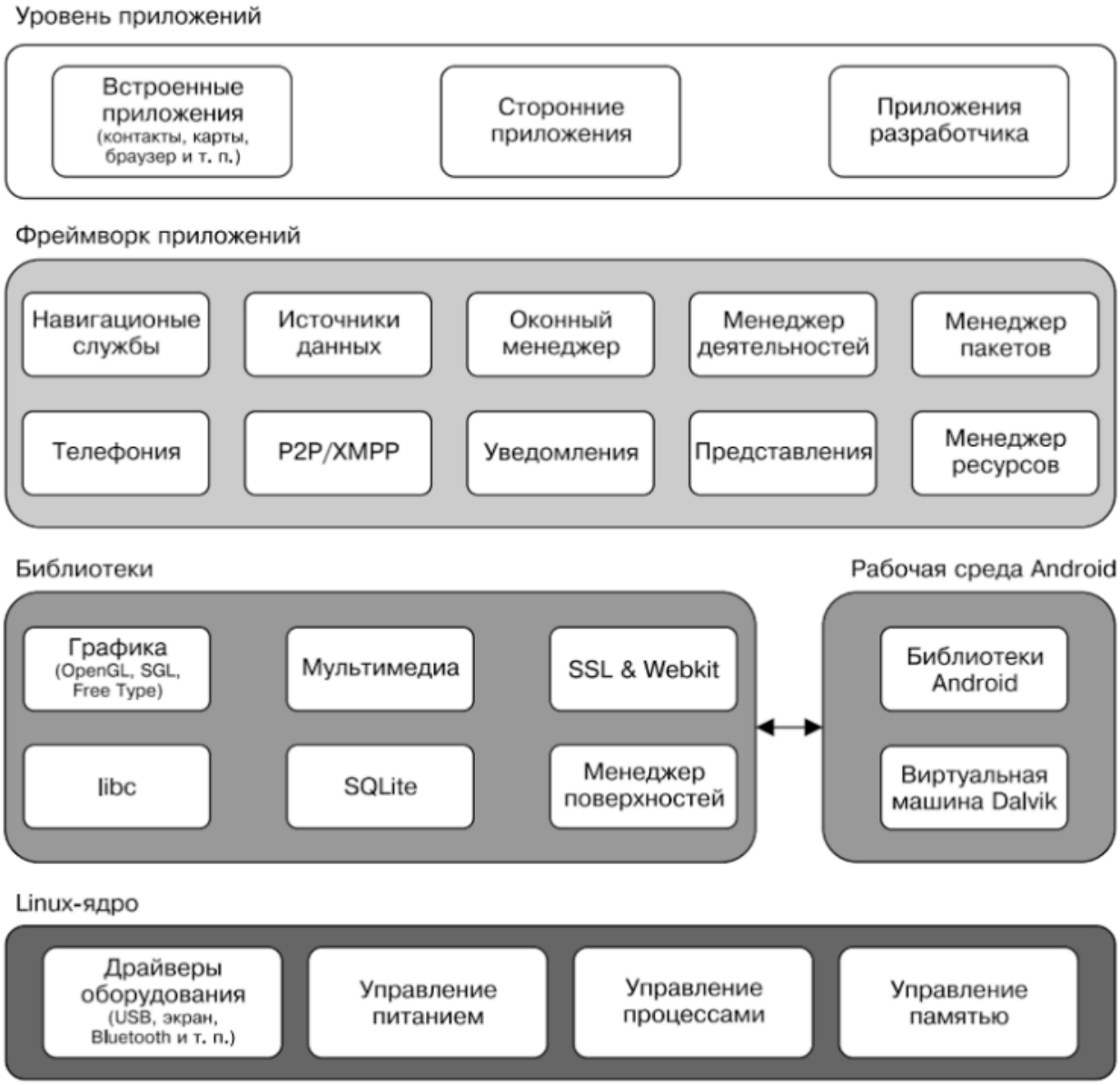


Рисунок 1.1 – Архитектура мобильной платформы Android

Ядро ***Linux*** обеспечивает функционирование системы и отвечает за безопасность, управление памятью, энергосистемой и процессами, а также предоставляет сетевой стек и модель драйверов. Набор библиотек (***Libraries***) предназначен для обеспечения важнейшего базового функционала приложений и отвечает за поддержку файловых форматов, кодирование и декодирование информации (например, цифровой звук и видео), отображение графики, поддержку ***Web***-компонентов (***WebKit***), ***SQL*** СУБД (***SQLite***) и стандартной для ***Linux***-систем функциональность библиотек ***C.*** На этом же уровне располагается рабочая среда ***Android (Android Runtime).*** Каждое приложение в операционной системе ***Android*** запускается в собственном экземпляре виртуальной машины ***Dalvik***. Таким образом, все работающие процессы изолированы от операционной системы и друг от друга. Благодаря этому осуществляется защита ядра операционной системы от возможного вреда со стороны других её составляющих.

Уровень каркаса приложений ***(Application Framework)*** включает основные службы ***Android*** для управления жизненным циклом приложений, пакетами, ресурсами и т. д. Программист имеет полный доступ к тем же ***API (Application Programming Interface),*** которые используются основными приложениями. Архитектура этих приложений разработана с целью упрощения многократного использования компонентов. Любое разрабатываемое приложение может использовать возможности базовых приложений и, соответственно, любое другое стороннее приложение может использовать возможности вашего приложения (с учетом установленных разрешений).

Уровень приложений (***Applications***) включает стандартные приложения ***Android*** (браузер ***WebKit***, календарь ***Google***, клиент ***Gmail***, приложение ***Gmaps***, ***SMS***-мессенджер и ***e-mail*** клиент), а также дополнительно загруженные приложения (из магазина ***Android***). ***Android*** не делает разницы между основными приложениями и сторонним программным обеспечением, в связи с чем ключевые компоненты, такие как набор номера, рабочий стол или почтовый клиент ***GMail***, можно заменить альтернативными аналогами.

Всего в ***Android*** приложениях существует четыре типа компонентов из уровня каркаса приложений: деятельность (***Activity***), служба (***Service***), приемник широковещательных намерений (***Broadcast Receiver***), контентпровайдер (***Content Provider).*** Взаимодействие компонентов осуществляется с помощью объектов ***Intent***.

Activity представляет собой визуальный пользовательский интерфейс для приложения – окно. ***Activity*** может также использовать дополнительные окна, например всплывающее диалоговое окно. Все деятельности реализуются как подкласс базового класса ***Activity***.

Компонент ***Service*** не имеет визуального интерфейса пользователя и выполняется в фоновом режиме в течение неопределенного периода времени, пока не завершит свою работу. ***Service***, как правило, требуется для длительных операций или для обеспечения работы удаленных процессов, но в общем случае это просто режим, который функционирует, когда приложение не в фокусе. Примером такого процесса может стать прослушивание музыки в то время, когда пользователь делает что-то другое или получение данных по сети без блокирования текущей активности. Приложения могут подключаться к компоненту ***Service*** или запускать его, если он не запущен, а также останавливать уже запущенные компоненты.

Broadcast Receiver используется для отслеживания внешних событий и реакции на них. Приложение может иметь несколько компонентов ***Broadcast Receiver***, чтобы ответить на любые объявления, которые оно считает важными. При этом каждый компонент ***BroadcastReceiver*** будет определять код, который выполнится после возникновения конкретного внешнего события. С помощью класса ***NotificationManager*** можно сообщить пользователю информацию, требующую его внимания. Примером оповещений может быть сигнал о том, что информация загружена на устройство и доступна к использованию.

Content Provider делает определенный набор данных, используемых приложением, доступным для других приложений. Данные могут быть сохранены в файловой системе, в базе данных ***SQLite***, в сети, или в любом другом месте, к которому приложение может иметь доступ. Контент-провайдеры для безопасного доступа к данным используют механизм разрешений. Посредством ***content provider*** другое приложение может запрашивать данные и, если выставлены соответствующие разрешения, изменять их. Например, система ***Android*** содержит ***content provider,*** который управляет пользовательской информацией о контактах.

Intent – специальные классы в коде приложения, которые определяют и описывают запросы приложения на выполнение каких-либо операций. Намерения добавляют слой, позволяющий оперировать компонентами с целью их повторного использования и замещения. В некоторых случаях это может быть очень мощным средством интеграции приложений. Главная

особенность платформы ***Android*** состоит в том, что одно приложение может использовать элементы других приложений при условии, что эти приложения разрешают использовать свои компоненты. При этом ваше приложение не включает код другого приложения или ссылки на него, а просто запускает нужный элемент другого приложения.

Создание проекта в ***Android Studio***.

Разработку приложений для платформы ***Android*** выполняют на языке ***Java***. С 2013 года разработка выполняется в официальной среде компании ***Google*** – ***Android Studio***, основанной на ***IntelliJ IDEA*** от ***JetBrains***.

Для создания нового проекта в ***Android Studio*** выбираем в окне приветствия опцию ***New Project*** (для уже открытого проекта в меню ***File*** следует выбрать ***New Project***). В появившемся окне ***Create New Project*** (рисунок 1.2) заполняем поля имени приложения ***(Application name***), которое будет отображается для пользователей, и квалификатора (***Company Domain***), который будет добавляться к имени пакета. Полное название проекта (***Package name***) формируется в соответствии с правилами именования пакетов в ***Java***. При необходимости изменяем папку размещения проекта (***Project location***).

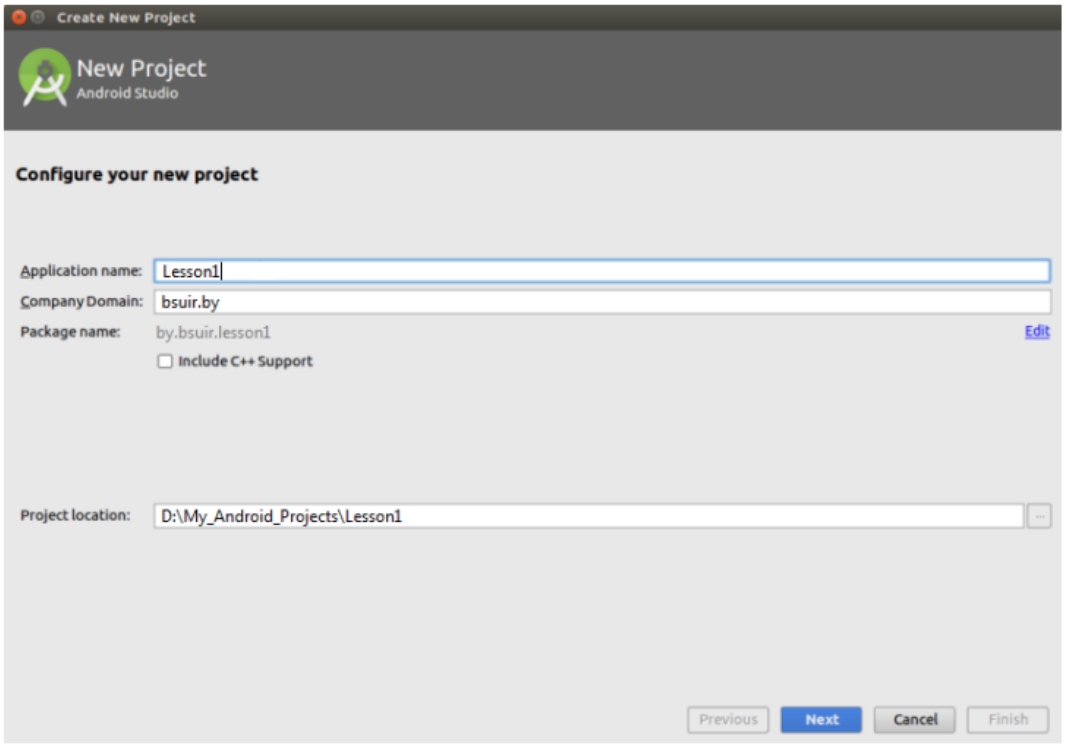


Рисунок 1.2 – Конфигурирование нового проекта в Android Studio

Далее (рисунок 1.3) указываем минимальную версию ***SDK*** платформы – самую раннюю версию ***Android***, которую будет поддерживать приложение.

Обратите внимание, что ***Minimum SDK*** не может быть выше версии ***API*** той мобильной платформы, на которой планируется тестирование и последующее использование приложения.

В появившемся окне ***Add an activity to Mobile*** в качестве основы будущего приложения выбираем ***Empty Activity***, в результате чего будет создано приложение с одним ***Activity***.

В форме ***Customize the Activity*** сгенерированы имена файлов класса ***Activity (Activity Name),*** графической разметки (***Layout Name***). Рекомендуется оставить данные имена без изменений, т.к. для запускающего ***Activity*** они являются общепринятыми.

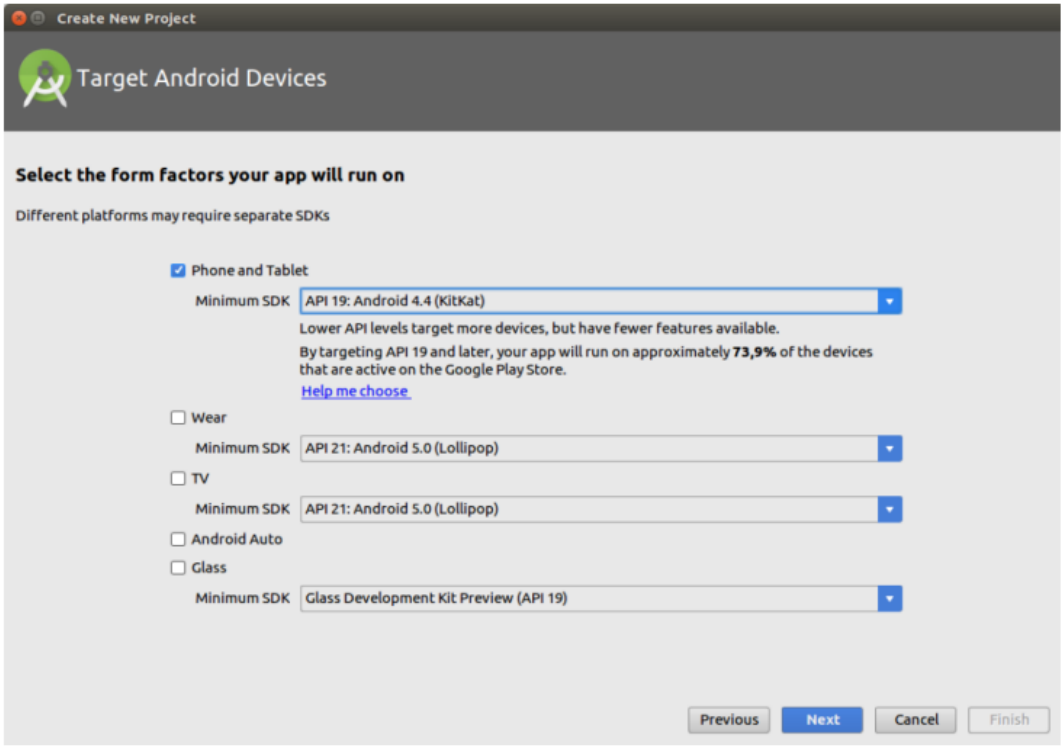


Рисунок 1.3 – Указание минимальной версии ***SDK*** для нового проекта в ***Android Studio***

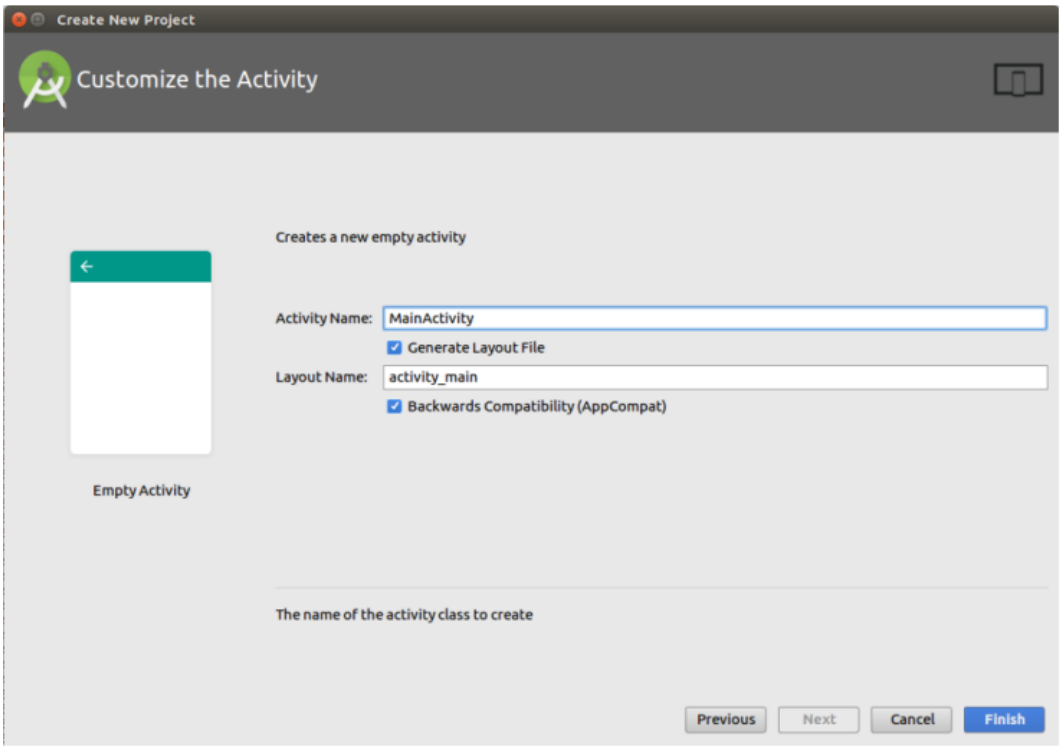


Рисунок 1.4 – Кастомизация имен для нового проекта в ***Android Studio***

После этого ***Android Studio*** создает новый проект с указанными характеристиками. Структура проекта (рисунок 1.5) включает: файл ***AndroidManifest.xml*** (манифест приложения), папку ***java*** (содержит весь код приложения), папку ***res*** (используется для файлов-ресурсов различного типа: ***res/drawable/*** - для изображений (***PNG, JPEG*** и т. д.); ***res/layout/*** - для ***xml*** файлов графической разметки; ***res/menu/*** - для ***xml***-файлов меню; ***res/values/*** - для строковых ресурсов, стилей и т. д.).

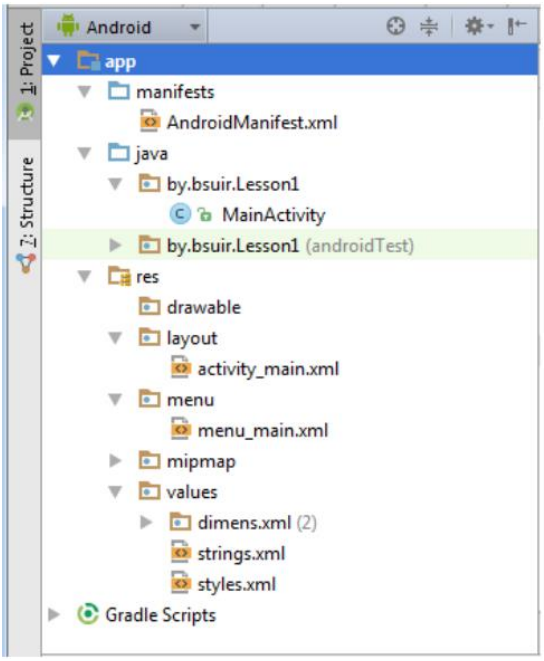


Рисунок 1.5 – Структура проекта в ***Android Studio***

Графическое представление ***Activity***.

Основу ***Android***-приложения составляет ***Activity*** – рабочее окно. В конкретный момент времени обычно отображается одно ***Activity*** и занимает весь экран. Работа с набором окон осуществляется путем переключения между различными ***Activity***. В качестве примера можно рассмотреть почтовое приложение: в нем одно ***Activity*** – список писем, другое – просмотр письма, третье – настройки ящика.

С программной точки зрения ***Activity*** представлено файлом графической разметки (***activity_main.xml*** на рисунке 1.5) и соответствующим ***java***-классом (***MainActivity*** на рисунке 1.5), реализующем запуск данного ***Activity*** с требуемым графическим представлением и последующей обработкой событий.

Графическое представление ***Activity*** формируется из различных компонентов (кнопка, поле ввода, чекбокс и т.д.), называемых ***View*** (рисунок 1.6).

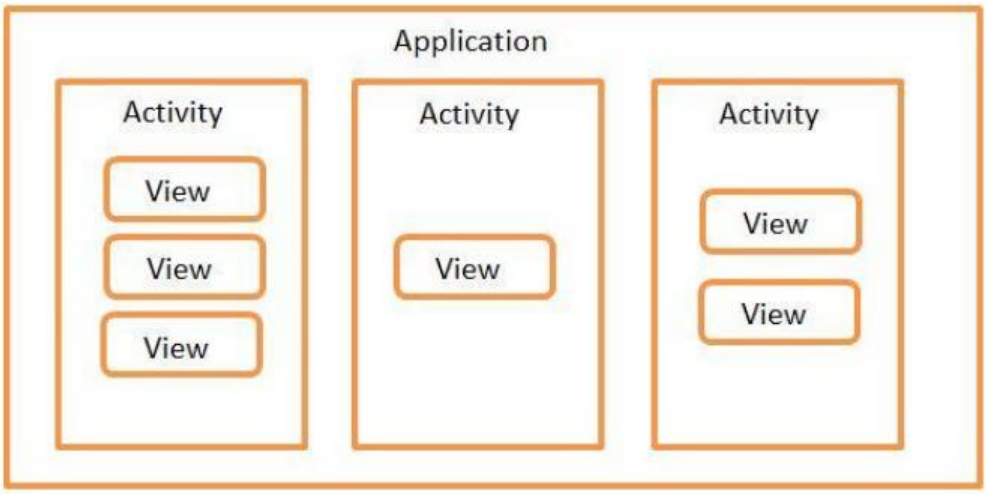


Рисунок 1.6 – Представление приложения ***Android*** как набора окон (***Activity***) с ***View***-компонентами Графическое представление каждого ***Activity*** в виде требуемого набора и взаимного расположения ***View***-элементов хранится в ***xml***-файле папки ***layout***. Данный файл графического представления прописывается в соответствующем ***java***-классе. При запуске приложения ***Activity*** читает этот файл и отображает его содержимое.

После создания проекта файл ***activity_main.xml*** открывается по умолчанию (рисунок 1.7) в режиме конструктора (вкладка ***Design***), помимо

которого также существует соответствующее ***xml***-описание графического представления (вкладка ***Text***).

На вкладке ***Design*** слева отображен список всех возможных виджетов (***View***-элементов), разделенный на группы. Справа на вкладке ***Component Tree*** отображаются все выбранные элементы, которые описаны в этом ***layout*** файле. Вкладка ***Properties*** отображает набор свойств для каждого выбранного элемента.

View-компоненты обычно группируют в ***ViewGroup***, называемые ***Layout***. ***Layout*** бывают различных типов и отвечают за то, как будут расположены их дочерние ***View***-компоненты на экране (таблицей, строкой, столбцом). Рассмотрим основные виды ***Layout***.

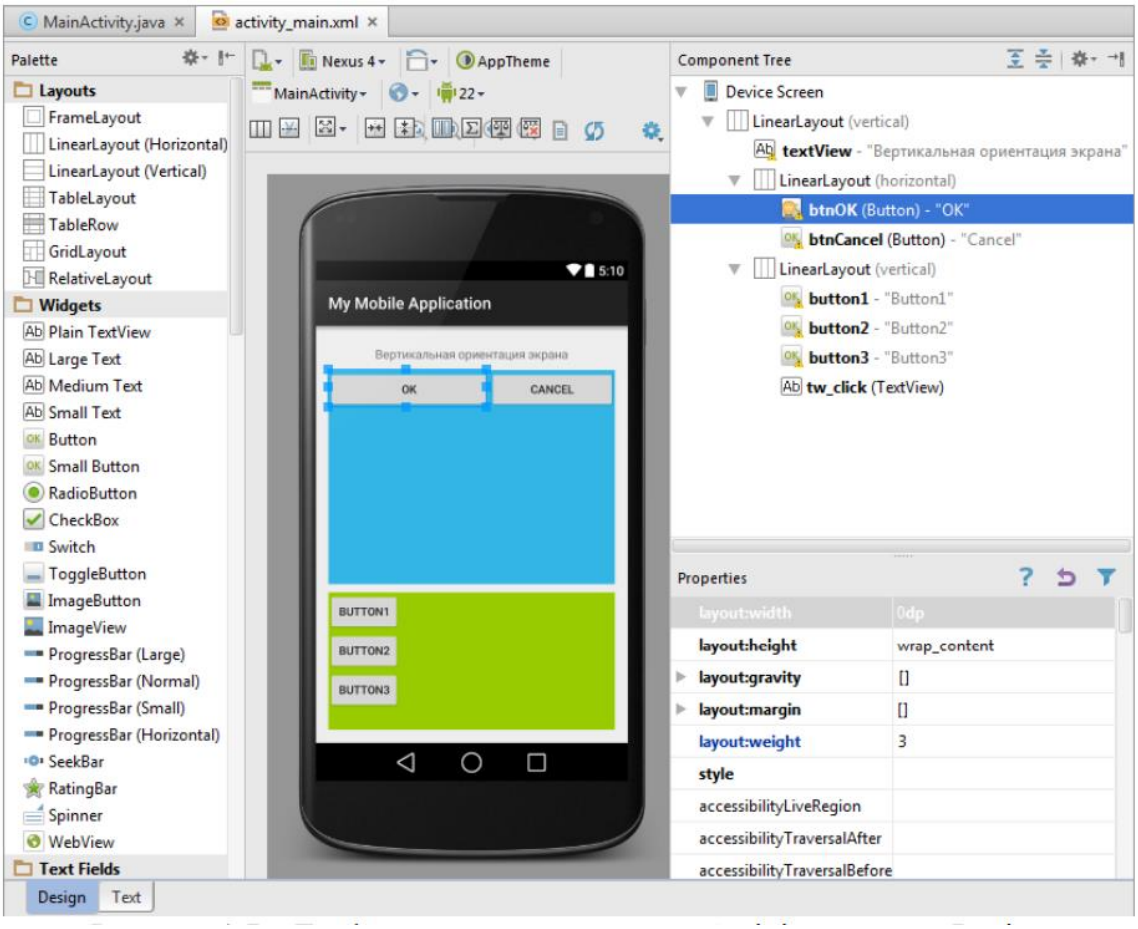


Рисунок 1.7 – Графическое представление ***Activity***, вкладка ***Design***

LinearLayout (LL) отображает ***View***-элементы в виде одной строки (***Horizontal***) или одного столбца (***Vertical***). Он удобен и достаточно гибок, чтобы создавать экраны различной сложности. ***LL*** имеет свойство ***Orientation***, которое определяет, как будут расположены дочерние элементы – горизонтальной или вертикальной линией.

TableLayout (***TL***) отображает элементы в виде таблицы. ***TL*** состоит из строк ***TableRow (TR),*** каждая ***TR*** в свою очередь содержит ***View***-элементы, формирующие столбцы. Т.е. кол-во ***View*** в ***TR*** - это кол-во столбцов. Но количество столбцов в таблице должно быть равным для всех строк. Поэтому, если в разных ***TR*** разное количество ***View***-элементов (столбцов), то общее количество определяется по ***TR*** с максимальным кол-вом.

RelativeLayout (RL) предусматривает настройку для каждого элемента его положения относительно других элементов. Виды отношений:

1) слева, справа, сверху, снизу указанного элемента (***layout_toLeftOf, layout_toRightOf, layout_above, layout_below).***

2) выравненным по левому, правому, верхнему, нижнему краю указанного элемента ***(layout_alignLeft, layout_alignRight, layout_alignTop,*** *layout_alignBottom).*

3) выравненным по левому, правому, верхнему, нижнему краю родителя (layout_alignParentLeft, layout_alignParentRight, layout_alignParentTop, layout_alignParentBottom).

4) выравненным по центру вертикально, по центру горизонтально, по центру вертикально и горизонтально относительно родителя ***(layout_centerVertical, layout_centerHorizontal, layout_centerInParent).***

AbsoluteLayout (AL) обеспечивает абсолютное позиционирование элементов на экране путем указания координат для левого верхнего угла компонента. Использование ***AL*** оправдано в случае разработки для экрана с конкретным разрешением. Если открыть такое приложение на другом экране, все элементы сместятся. Поэтому в общем случае ***AL*** не рекомендуется использовать. Его совместимость с будущими версиями ***Android*** не гарантируется.

Путем вложения в один ***Layout*** других ***Layout*** можно создавать экраны различной сложности. В рамках одного ***Layout*** располагают требуемые ***View*** элементы и настраивают их параметры.

Рассмотрим параметры ***View***-элементов.

Параметр ***id*** – это ***ID*** элемента. Чтобы обратиться к элементу экрана из кода, потребуется его ***ID***. Для ***ID*** существует четкий формат: ***@+id/name,*** где **+** означает, что это новый ресурс и он должен добавиться в ***R.java*** класс, если он там еще не существует. Параметры ***layout_width*** (ширина элемента) и

layout_height (высота элемента) могут быть следующими: ***fill_parent*** или ***match_parent*** (максимально возможная ширина или высота в пределах родителя), ***wrap_content*** (ширина или высота определяется по содержимому элемента), а могут задаваться в значениях:

dp (Density-independent Pixels) – абстрактная единица измерения, позволяющая приложениям выглядеть одинаково на различных экранах и разрешениях;

sp (Scale-independent Pixels) – то же, что и dp, только используется для размеров шрифта в View элементах;

pt – 1/72 дюйма, определяется по физическому размеру экрана;

px – пиксел, не рекомендуется использовать т.к. на разных экранах приложение будет выглядеть по-разному;

mm – миллиметр, определяется по физическому размеру экрана;

in – дюйм, определяется по физическому размеру экрана.

Если родитель содержит несколько элементов и необходимо, чтобы они заняли все пространство, целесообразно использовать параметр ***Layout weight*** – вес. Свободное пространство распределяется между элементами пропорционально их ***weight***-значениям. При использовании ***weigth*** вы можете указать значение ***height*** или ***width = 0dp.*** В этом случае не будет учитываться содержимое элементов и результате будет более соответствующий коэффициентам веса.

Параметр ***Layout gravity*** – это выравнивание. Параметр ***Layout margin*** – это отступ. При переходе на вкладку ***Text*** отображается ***xml***-описание всех элементов ***layout***-файла (рисунок 1.8).

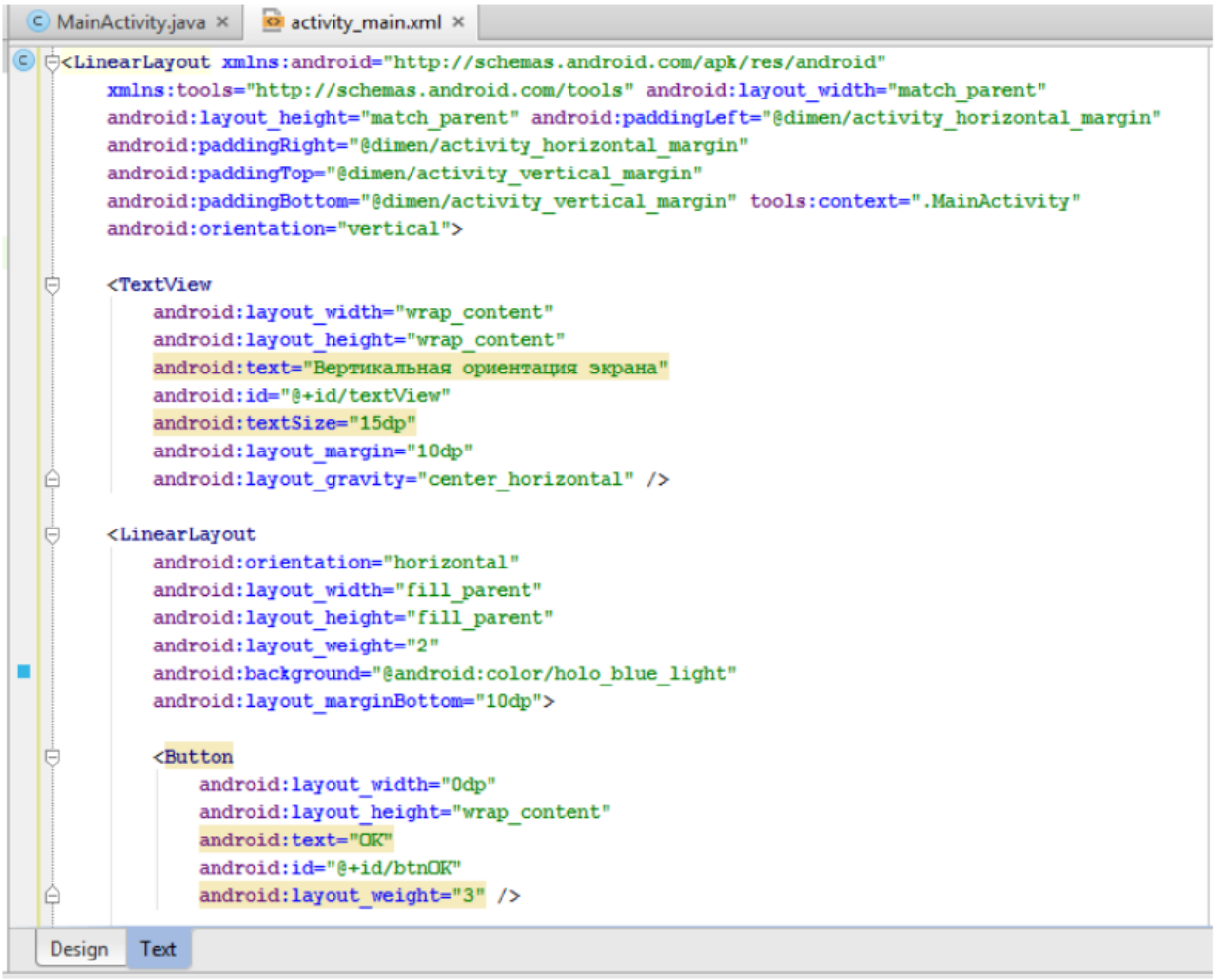
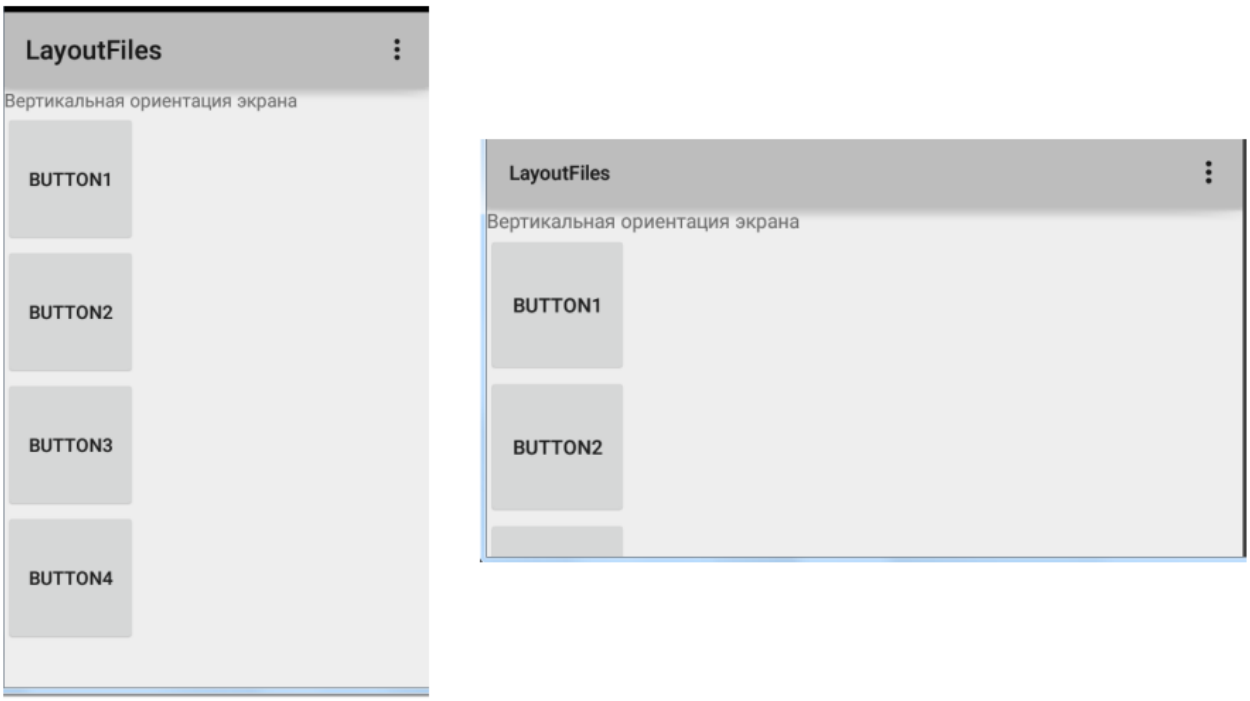


Рисунок 1.8 – Графическое представление ***Activity***, вкладка ***Text***

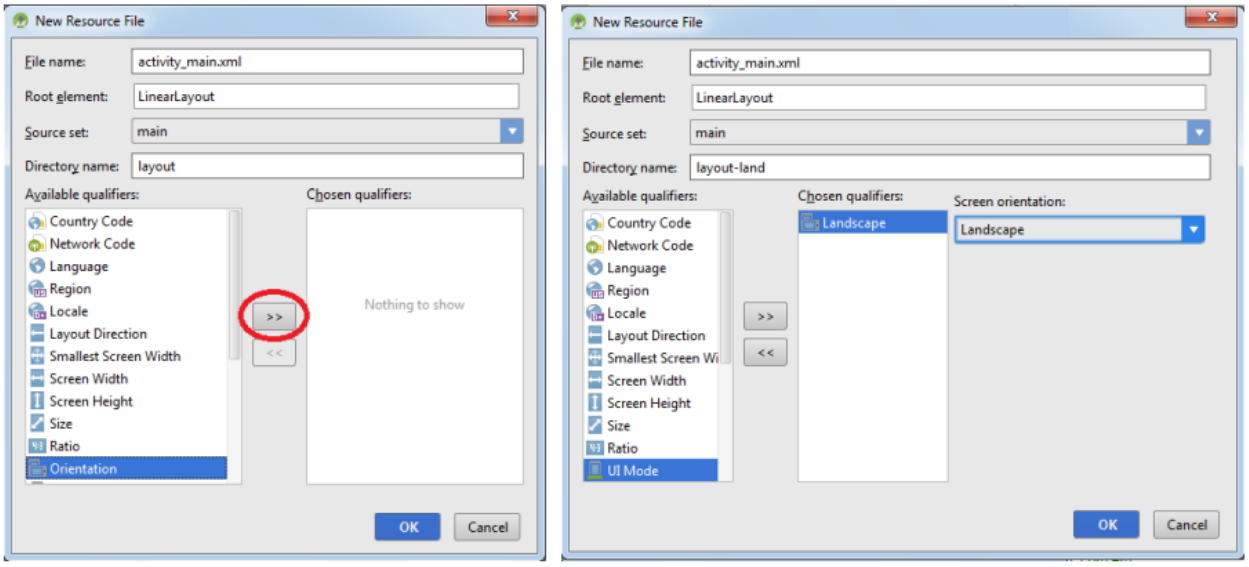
Названия ***xml***-элементов - это классы ***View***-элементов, ***xml***-атрибуты - это параметры ***View***-элементов, т.е. все те параметры, которые можно менять через вкладку ***Properties***. Также можно вносить изменения прямо в ***xml***-код, соответствующие изменения будут отображаться и во вкладке ***Design. Layout***-файл при смене ориентации экрана. По умолчанию мы настраиваем ***layout***-файл под вертикальную ориентацию экрана. Однако при повороте смартфона включится горизонтальная ориентация, что может привести к некорректному отображению ***View***-элементов (рисунок 1.9).



а б

Рисунок 1.9 – Пример некорректного отображения ***View***-элементов при смене ориентации экрана с вертикальной (а) на горизонтальную (б)

Для устранения данной проблемы необходимо создать еще один ***layout*** файл для горизонтальной ориентации экрана. В папке ***res/layout*** создаем новый ***Layout resource file***. Название файла указываем то же, что и для ***xml***-представления в вертикальной ориентации. В списке квалификаторов слева находим ***Orientation*** (рисунок 1.10, а) и нажимаем кнопку со стрелкой вправо. Далее выбираем параметр ***Landscape*** из выпадающего списка (рисунок 1.10, б). Обратите внимание на автоматическое изменение значения поля ***Directory name***.



а б

Рисунок 1.10 – Создание графического представления для горизонтальной ориентации экрана: выбор квалификатора ***Orientation*** (а) и установка параметра ***Landscape*** (б)

В структуре проекта появится новый ***xml***-файл ***activity_main.xml(land).*** Копируем в него содержимое файла ***activity_main.xml*** и меняем параметры ***layout*** и ***view***-элементов таким образом, чтобы в горизонтальной ориентации экрана они отображались корректно. При этом ***id*** элементов не меняем!

При запуске приложения ***Activity*** прочитает подключенный в коде ***layout***-файл и отобразит его содержимое. При этом будет учтена ориентация экрана, и в случае горизонтального расположения автоматически отобразится файл ***land***.

Файл ***MainActivity.java.*** Подключение графического представления к ***Activity***.

При запуске деятельности система должна получить ссылку на корневой узел дерева разметки, который будет использоваться для прорисовки графического интерфейса на экране мобильного устройства. Для этого в методе ***onCreate( )*** необходимо вызвать метод ***setContentView( ).***

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

Метод ***setContentView(int)*** – устанавливает содержимое ***Activity*** из ***layout***-файла. Но в качестве аргумента указывается не путь к ***layout***-файлу ***(res/layout/activity_main.xml),*** а константа, которая является ***ID*** файла и хранится в файле ***R.java.*** Имена этих ***ID***-констант совпадают с именами файлов ресурсов (без расширений).

Можно создать новый ***xml***-файл в папке ***res > layout*** и прописать его вместо ***activity_main*** в методе ***setContentView(int).*** После запуска приложения отобразится новый файл разметки.

Доступ к элементам экрана из кода.

Чтобы обратиться к элементу экрана из кода, необходим его ***ID***. Он прописывается в ***Properties*** либо в ***layout***-файлах:

```
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="OK"
android:id="@+id/btnOK" />
```

Важно давать уникальные и осмысленные ***ID***-имена элементам экрана! Зная ***ID View***-элемента, обратиться к нему из кода можно по константе ***R.id.btnOK***. Для этого понадобится метод ***findViewById:***

```
public class MainActivity extends ActionBarActivity {
    private Button btnOK;

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

    private void initViews() {
        // находим View-элементы
        btnOK = (Button) findViewById(R.id.btnOK);
    }
}
```

Следует отметить, что в вышеприведенном фрагменте кода обнаружение ***View***-элементов вынесено в отдельный пользовательский (не являющийся библиотечным) метод ***initViews***() для реализации принципа модульного программирования.

Обработка событий на примере нажатия кнопки.

Механизм обработки нажатия на кнопку основан на использовании интерфейса ***View.OnClickListener*** и реализации метода ***onClick***, в котором и прописывается логика действий в ответ на нажатие (рисунок 1.11).

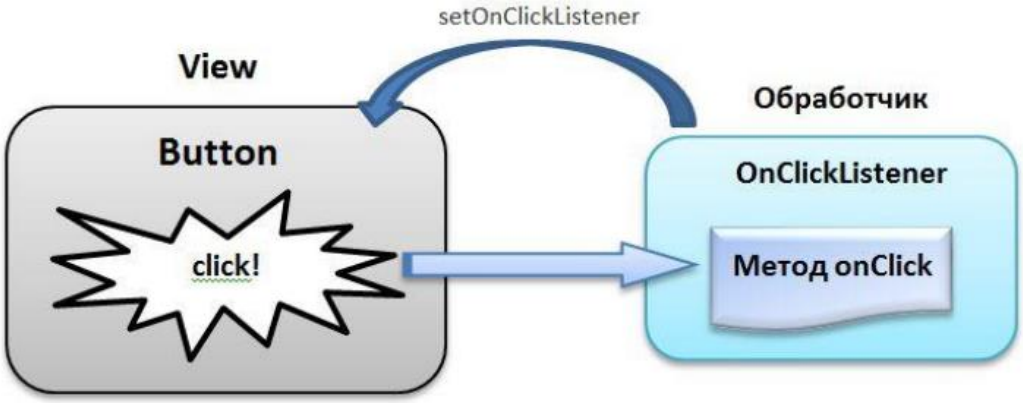


Рисунок 1.11 – Механизм обработки нажатия на кнопку на основе интерфейса ***View.OnClickListener***

При этом обработчик (его также называют слушателем - ***listener***) присваивается кнопке с помощью метода s***etOnClickListener***. Когда на кнопку нажимают, обработчик реагирует и выполняет код из метода ***onClick***. Для реализации вышеописанного механизма необходимо выполнить следующие шаги:

– создать обработчик (объект от интерфейса ***View.OnClickListener***), – заполнить метод ***onClick*** (т.к. в интерфейсе он не реализован, а только заявлен),

– присвоить обработчик кнопке (используем метод ***setOnClickListener***).

Система обработки событий готова: когда на кнопку нажимают, обработчик реагирует и выполняет код из метода ***onClick***.

Пример:

```
OnClickListener listener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        // метод, который будет вызван по нажатию
        }
};
button.setOnClickListener(listener);
```

Существуют различные способы программной реализации обработки нажатий от требуемого набора кнопок одного ***Activity*** (с использованием в ***xml***-представлении атрибута ***onClick***, своего обработчика для каждого ***View*** элемента, одного обработчика для нескольких ***View***-элементов). Однако наиболее оптимальным является использование ***Activity*** в качестве единого

обработчика. В данном случае сам класс ***Activity*** реализует интерфейс ***View.OnClickListener***:

```
public class MainActivity extends ActionBarActivity
implements View.OnClickListener{
    private Button btnOK, btnCancel;
    @Override
    protected void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initViews();
    }
    private void initViews() {
        // находим View-элементы
        btnOK = (Button) findViewById(R.id.btnOK);
        btnCancel = (Button)
findViewById(R.id.btnCancel);
        // подключаем обработчик к кнопкам
        btnOK.setOnClickListener(this);
        btnCancel.setOnClickListener(this);
    }
    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.btnOK: btnOK.setText("Hello");
                btnOK.setEnabled(false);
                btnCancel.setEnabled(true);
            break;
            case R.id.btnCancel:
                btnCancel.setText("Goodbye");
                btnCancel.setEnabled(false);
                btnOK.setEnabled(true); break;
        }
    }
}
```

Для считывания данных из текстовых полей (например, из поля ***et_message***) используется следующий код:

```
String message = et_message.getText().toString();
```

Практическое задание

1. Разработать приложение ***Calculator*** с одним ***Activity***.

2. Графическое представление приложения ***Calculator*** реализовать с кнопками цифр, математических операций (сложения, вычитания, умножения, деления и др.), получения результата (пример приведен на рисунке 1.12, а).

3. Предусмотреть возможность ввода дробных чисел через точку (например, 0.5).

4. Предусмотреть возможность ввода отрицательных чисел.

5. Создать графическое представление приложения ***Calculator*** для горизонтальной ориентации экрана (пример приведен на рисунке 1.12, б).

6. Программно реализовать обработку нажатий на кнопки с использованием ***Activity*** в качестве единого обработчика.

7. В случае деления на 0 выводить вместо результата сообщение о недопустимости операции.

8. Разработать перечень проверок и протестировать приложение ***Calculator***.

9. Продемонстрировать работу приложения ***Calculator*** на эмуляторе или реальном устройстве. а б Рисунок 1.12 – Пример работы приложения Калькулятор в вертикальной (а) и горизонтальной (б) ориентации экрана

Содержание отчета

1. Скриншоты графических представлений приложения ***Calculator*** в вертикальной и горизонтальной ориентациях экрана в ***Android Studio.***

2. Код ***xml***-файла графического представления приложения ***Calculator*** в вертикальной ориентации экрана.

3. Код ***xml***-файла графического представления приложения ***Calculator*** в горизонтальной ориентации экрана.

4. Код ***java***-файла Activity приложения ***Calculator***.

Контрольные вопросы

1. Что такое нативное мобильное приложение, мобильная платформа?

2. Что собой представляет архитектура мобильной платформы ***Android***?

3. Какие основные компоненты ***Android***-приложения Вы знаете?

4. Что собой представляет структура ***Android***-проекта? Что содержит файл конфигурации ***AndroidManifest.xml***, папка ***java***, папка ***res***?

5. Что такое графическое представление ***Activity***?

6. Что такое ***Layout***? Какие существуют виды ***Layout***?

7. Какие параметры (атрибуты) имеют ***View***-элементы?

8. Как создать ***Layout***-файл для работы в горизонтальной ориентации экрана мобильного устройства? В каких случаях это необходимо?

9. Для чего нужны методы ***setContentView, findViewById***?

10. Какие существуют способы обработки событий в ***Activity***?