МИНИСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО «Кубанский государственный
аграрный университет имени И. Т. Трубилина»

Е. А. Иванова, Т. А. Крамаренко

КРОССПЛАТФОРМЕННЫЕ ПРИЛОЖЕНИЯ

Практикум

Краснодар
КубГАУ
2020

УДК 004.43 (075.8)
ББК 32.973
И20

Иванова Е. А.

И20 Кроссплатформенные приложения : практикум / Е. А. Иванова, Т. А. Крамаренко. – Краснодар : КубГАУ, 2020. – 172 с.

Представлены теоретические учебные и методические материалы для выполнения практических заданий и лабораторных работ по дисциплине «Кроссплатформенные приложения». Также содержит примеры выполнения заданий и требования к оформлению отчетов.

Предназначен для студентов высших учебных заведений, обучающихся по направлению подготовки 09.03.02 Информационные системы и технологии всех форм обучения.

Предисловие

Дисциплина «Кроссплатформенные приложения» является одной из ключевых в учебном плане подготовки обучающихся по направлению 09.03.02 Информационные системы и технологии.

Практикум по дисциплине «Кроссплатформенные приложения» содержит практические задания и лабораторные работы для получения практических навыков:

– разработки мобильных кроссплатформенных приложений в среде Android Studio;

– визуального дизайна интерфейсов мобильных приложений с применением различных контейнеров компоновки и элементов управления;

– программирования игровых приложений для смартфонов;

– создания и модификации многооконных приложений, а также приложений, содержащих несколько активностей;

– использования в приложениях отличительных возможностей смартфонов;

– подключения и использования библиотек в мобильных кроссплатформенных приложениях;

– реализации графики и анимации в мобильных приложениях.

Кроме того, практикум содержит примеры выполнения заданий, а также требования к оформлению отчетов.

Практическая работа №1. Установка и настройка Android Studio

1 Цель работы

Получить практические навыки установки и настройки Android Studio, тестирования приложений на эмуляторах и реальных устройствах.

2 Порядок выполнения работы

2.1 Установка Android Studio

Android – это бесплатная операционная система, основанная на Linux с интерфейсом программирования Java.

Важно понимать, что само приложение пишется на Java, а среда разработки выбирается по желанию. То, что можно сделать на Android Studio, можно сделать на Eclipse, либо в других редакторах. Хотя с 2016 официальная поддержка Eclipse прекратилась.

Скачать установочный пакет для студии можно со страницы http://developer.android.com/sdk/index.html

Сама установка проблем осуществляется достаточно легко. Установочный пакет включает в себя необходимый минимум. Иногда необходимо запускать Android SDK Manager и проверять наличие новых версий SDK через меню **Tools | Android | SDK Manager** (рисунок П.1.1)**.**

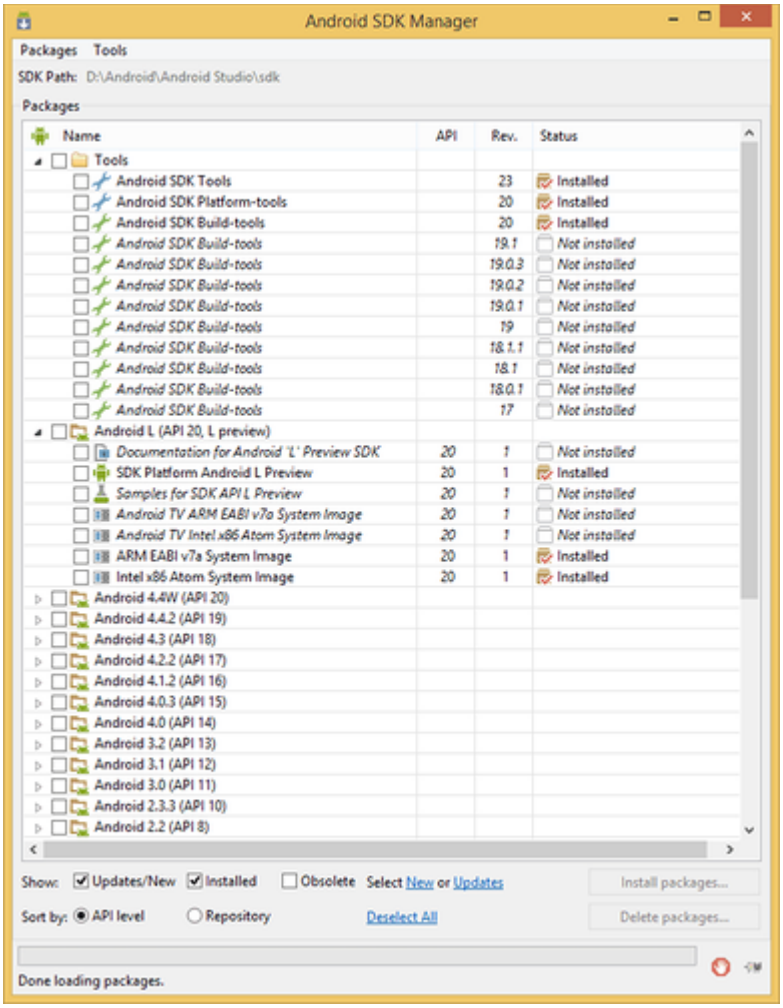

Рисунок П.1.1 – Проверка наличия новых версий SDK

Со временем приходит понимание, что нужно ставить, а что не обязательно. На первых порах можно согласиться на те условия, которые предложит менеджер.

В некоторых случаях в Win64 требуются права администратора при установке или обновлении.

В версии студии 2.3 или раньше ставится пакет OpenJDK, который является альтернативой JDK от Oracle. В настройках студии есть примечание, что OpenJDK является рекомендуемым вариантом, хотя можно указать путь и к стандартной JDK. В любом случае будет установлен Java 8 вне зависимости, какой вариант будет выбран.

2.2 Создание эмуляторов

Для отладки приложений используется эмулятор телефона - виртуальная машина, на которой будет запускаться приложение. Также можно использовать и реальное устройство.

В момент установки студия создаёт одно устройство для эмулятора. Если это не так, его всегда можно установить вручную. А также можно добавить и другие устройства под разные версии Android, разрешения экрана и т.д.

Чтобы создать эмулятор телефона, выбираем в меню **Tools | Android | AVD Manager**. При первом запуске появится диалоговое окно мастера, представленное на рисунке П.1.2.

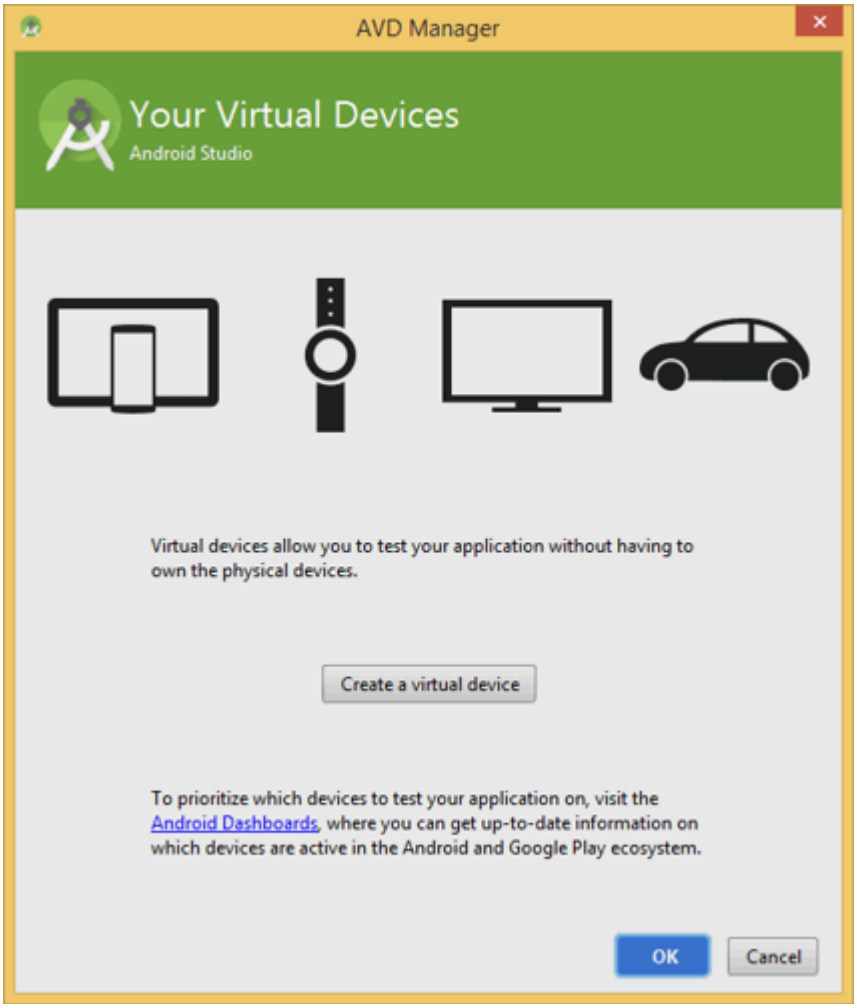

Рисунок П.1.2 – Мастер создания эмулятора

После нажатия кнопки **Create a virtual device** в новом окне видим набор возможных эмуляторов, в том числе и для часов (рисунок П.1.3). Необходимо скачать необходимые эмуляторы. Для начала вполне подойдёт один эмулятор.

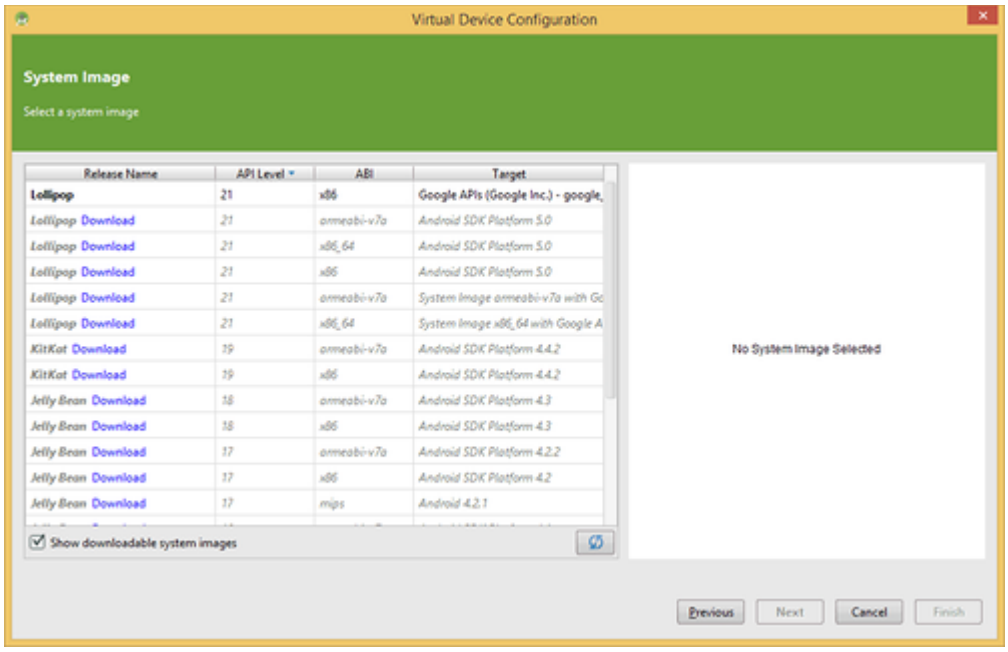

Рисунок П.1.3 – Список возможных эмуляторов

Вводим любое понятное имя, например, Android4, выбираем нужную версию Android, размер экрана и т.д.

При необходимости можно создать эмуляторы для каждой версии ОС и проверять программу на работоспособность. Остальные настройки можно оставить без изменений. Всегда можно вернуться к настройкам и отредактировать снова. Часто рекомендуют использовать опцию **Use Host GPU**, чтобы задействовать возможности графического процессора. Это даёт прирост скорости эмулятора. Нажимаем кнопку **OK**.

Добавленные эмуляторы будут храниться в менеджере эмуляторов (рисунок П.1.4).

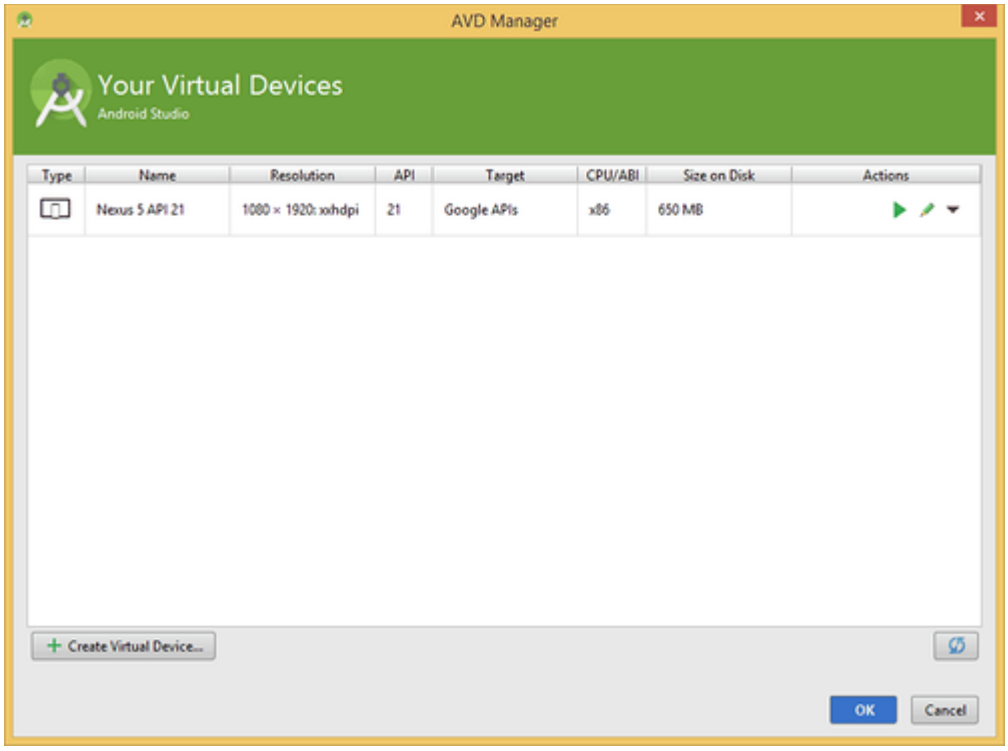

Рисунок П.1.4 – Менеджер эмуляторов

Если было создано несколько эмуляторов, то следует выделить нужный и нажать кнопку с зелёным треугольником для запуска эмулятора. Значок карандаша позволяет отредактировать настройки. Но обычно эмулятор не запускают отдельно. Когда разработчик будет запускать приложение, то студия сама предложит запустить эмулятор.

Помните, что виртуальные машины по умолчанию сохраняются в папке пользователя, и пути к папке не должны содержать русские символы во избежание проблем.

Если данная проблема все-таки произошла, то по данной ссылке можно ознакомиться с информацией по смене папки пользователя на английский язык: http://www.cherneenet.ru/lokalnaj_zapis.html. Также можно подправить ini-файл и прописать путь к виртуальному устройству таким образом, чтобы в пути не встречались русские буквы (соответственно, сам файл *.avd также нужно переместить в другое место).

В зависимости от мощности компьютера нужно немного подождать, чтобы сначала загрузился эмулятор.

2.3 Тестирование на реальном устройстве

Окончательную версию приложения желательно проверять на настоящем устройстве. Начиная с Android 4.4, на устройстве (планшет, телефон, часы) нужно активировать режим разработчика. Для этого следует перейти в **Настройки**, там открыть страницу **«О телефоне»** и щёлкнуть семь раз на строчке с номером сборки **Build number**. После этого в настройках появится новый пункт **Для разработчиков** (или похожий), если производитель использует свою оболочку.

Открыв страницу для разработчиков, нужно включить отладку через USB. Также можно включить опцию **Не выключать экран**. Для некоторых устройств требуется установить отдельный драйвер, ищите информацию на сайте у производителей.

В случае возникновения проблем последнюю версию документации можно найти по адресу: developer.android.com.

3 Контрольные вопросы

1. Для каких целей используется SDK Manager?
2. Что такое эмулятор? Как его создать?
3. Каким образом можно выбрать виртуальное устройство для запуска приложения?
4. Какие настройки необходимо выполнить, чтобы запустить приложение на реальном устройстве?

4 Задание для самостоятельного выполнения

Произведите установку Android Studio, протестируйте приложение «Hello, world!» на любом эмуляторе и реальном устройстве.

Практическая работа №2. Разработка простейшего приложения под Android

1 Цель работы

Получить практические навыки разработки простейших мобильных приложений с обработчиками событий.

2 Порядок выполнения работы

Прежде чем переходить к выполнению работы, следует учесть, что студия постоянно обновляется, поэтому внешний вид окон и другие детали могут отличаться от данного примера.

В качестве языка программирования для Android используется Java. Для создания пользовательского интерфейса используется XML.

Приложение «Hello World!» уже встроена в среду разработки под Android в целях совместимости, поэтому разобьём задачу на две части. Сначала запустим готовую программу **Hello World!** без написания кода, чтобы убедиться, что весь инструментарий корректно установился, и мы можем создавать и отлаживать программы. А потом уже напишем свою первую программу.

2.1 Создание нового проекта

Для этого следует запустить Android Studio и выбрать **File | New | New Project...**. Появится диалоговое окно мастера.

Поле **Application name:** – понятное имя для приложения, которое будет отображаться в заголовке приложения. По умолчанию уже может быть **My Application**. Заменяем на **Hello World**. В принципе здесь можно написать и **Здравствуй, мир!**, но у Android есть замечательная возможность выводить нужные строки на телефонах с разными языками. Скажем, у американца на телефоне появится надпись на английском, а у русского - на русском. Поэтому в первоначальных настройках всегда исполь-

зуются английские варианты, а локализованные строки готовятся позже. Необходимо сразу вырабатывать привычку к правильному коду.

Поле **Company Domain:** служит для указания сайта. По умолчанию там может появиться ваше имя как пользователя компьютера. Если сайт у вас есть, то можете ввести его адрес, либо придумайте какое-нибудь название. Введённое имя запоминается и будет автоматически подставляться в следующих новых проектах.

Поле **Package name:** формирует специальный Java-пакет на основе имени из предыдущего поля. В Java используется перевёрнутый вариант для наименования пакетов, поэтому сначала идёт **ru**, а потом уже название сайта. Пакет служит для уникальной идентификации приложения, когда оно будет распространяться. Обратите внимание, что Гугл в своей документации использует пакет **com.example** в демонстрационных целях. То есть, если просто копировать примеры из документации и в таком виде пытаться выложить их в Google Play, то ничего не получится – это название зарезервировано и запрещено к использованию в магазине приложений. Кнопка **Edit** позволяет отредактировать подготовленный вариант. Например, приложение пишется на заказ и нужно использовать имя пакета, утверждённое заказчиком, а не вариант по умолчанию.

Третье поле **Project location:** позволяет выбрать место на диске для создаваемого проекта.

Нажимаем на кнопку **Next** и переходим к следующему окну. Здесь происходит выбор типов устройств, под которые будет разрабатываться приложение. В большинстве случае это будут смартфоны и планшеты, поэтому оставляем флажок у первого пункта. Также можно писать приложения для Android TV, Android Wear и Glass.

Кроме выбора типа устройств, надо выбрать минимальную версию системы, под которую будет работать приложение.

Если щёлкнуть по ссылке Help me choose, то откроется окно с графиком.

Снова нажимаем кнопку **Next**. На данном этапе следует выбрать внешний вид экрана приложения (рисунок П.2.1).

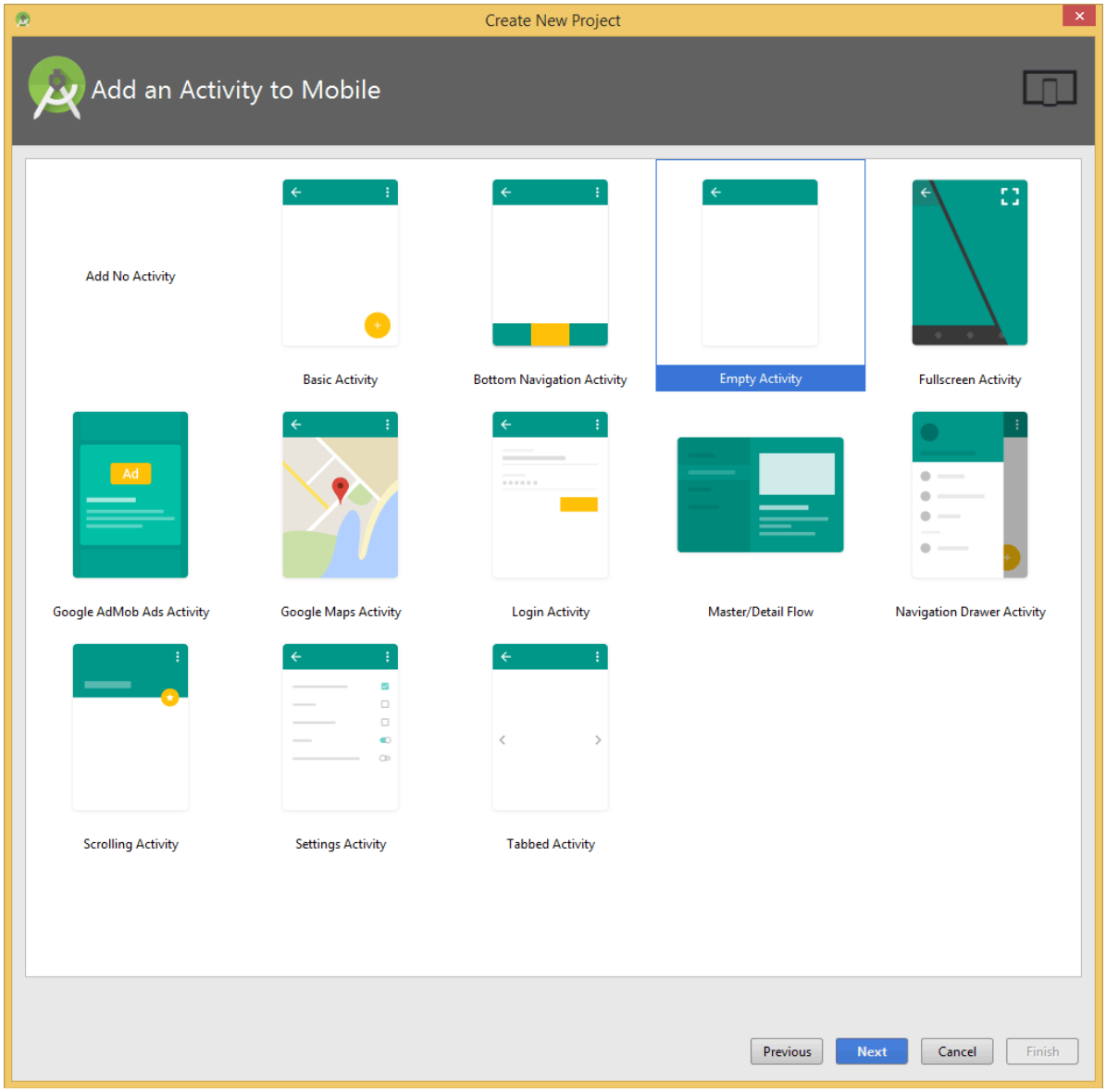

Рисунок П.2.1 – Выбор главного экрана приложения

Предложенные шаблоны позволяют сэкономить время на написание стандартного кода для типичных ситуаций. Опытный разработчик может вручную написать любой из предложенных вариантов, используя вариант **Add No Activity**, где никаких заготовок не будет.

Несколько лет назад использовался только один шаблон. Список шаблонов постоянно пополняется и теперь их уже достаточно много. Перечислим часть из них:

- **Basic Activity;**
- **Bottom Navigation Activity;**
- **Empty Activity;**
- **Fullscreen Activity;**
- **Google Maps Activity;**
- **Google AdMod Ads Activity;**
- **Login Activity;**
- **Master/Detail Flow;**
- **Navigation Drawer Activity;**
- **Scrolling Activity.**

Шаблон **Empty Activity** предназначен для обычных телефонов. На картинке над названием шаблона виден приблизительный вид приложения с использованием данной заготовки. Для учебных программ в 99% подойдёт этот вариант. Практически все дальнейшие примеры написаны с помощью данного шаблона.

Шаблон **Master/Detail Flow** предназначен для планшетов с реализацией двухпанельного режима. Шаблон **Fullscreen Activity** можно использовать для игр, когда требуется дополнительное пространство без лишних деталей. Другие шаблоны нужны для создания приложений с гугл-картами или сервисами Google Play.

Итак, выбираем вариант **Empty Activity** и переходим к следующему окну. Нажимаем кнопку **Finish**.

В результате студия формирует проект и создаёт необходимую структуру из различных файлов и папок.

Боковая левая часть студии имеет несколько вертикальных вкладок. Скорее всего, будет активна первая вкладка **1:Project**. Вкладки **Structure** и **Captures** используются гораздо реже.

В левой части среды разработки на вкладке **Android** появится иерархический список из папок, которые относятся к проекту. В некоторых случаях желательно пере-

ключиться на режим **Project**, который показывает истинное расположение файлов. Но на первых порах удобнее использовать именно вид **Android**, который прячет служебные файлы.

2.2 Содержание проекта

Вкладка **Android** содержит две основные папки: **app** и **Gradle Scripts**. Первая папка **app** является отдельным модулем для приложения и содержит все необходимые файлы приложения – код, ресурсы картинок и т. п. Вторая папка служит для различных настроек, управления проектом и многих других вещей.

На данный момент нас интересует папка **app**. В ней находятся три папки: **manifest**, **java**, **res**.

manifest

Папка **manifest** содержит единственный файл манифеста **AndroidManifest.xml**. В этом файле должны быть объявлены все активности, службы, приёмники и контент-провайдеры приложения. Также он должен содержать требуемые приложению разрешения. Например, если приложению требуется доступ к сети, это должно быть определено здесь. «AndroidManifest.xml» можно рассматривать, как описание для развертывания Android-приложения.

Более подробно о структуре манифеста можно прочитать здесь: Файл AndroidManifest.xml

java

Папка **java** содержит три подпапки – рабочую и для тестов. Рабочая папка имеет название пакета и содержит файлы классов. На текущем этапе там один класс **MainActivity**. Папки для тестов можете не использовать.

res

Папка **res** содержит файлы ресурсов, разбитых на отдельные подпапки:

- **drawable** – в этих папках хранят графические ресурсы – картинки и xml-файлы, описывающие цвет и фигуры.
- **layout** – в данной папке содержатся xml-файлы, описывающие внешний вид форм и различных элементов форм. После создания проекта там уже имеется файл **activity_main.xml**, который отвечает за внешний вид главного окна приложения.
- **mipmap** – здесь хранят значки приложения под разные разрешения экрана
- **values** – тут размещаются строковые ресурсы, ресурсы цветов, тем, стилей и измерений, которые можно использовать в проекте. Здесь находятся файлы **colors.xml**, **dimens.xml**, **strings.xml**, **styles.xml**

2.3 Работа с проектом Hello, World!

Как уже говорилось, программа **Hello, World!** уже встроена в любой новый проект, поэтому даже не нужно писать никакого кода. Просто нужно запустить проект и получить готовую программу.

Для его изучения нужно открыть два файла – **MainActivity** (скорее всего он уже открыт) и **activity_main.xml** (**res/layout**) в центральной части Студии. Если файлы не открыты, то откройте их самостоятельно двойным щелчком для редактирования (или просмотра). Таким способом можно открыть любой нужный файл.

Не будем пока изучать код, а просто нажмём на зелёный треугольник **Run** (Shift+F10) на панели инструментов в верхней части студии для запуска приложения.

Если все сделано правильно, то в эмуляторе или на устройстве загрузится программа (рисунок П.2.2).

В результате появится окно приложения с надписью **Hello World**. Заголовок у программы будет также **Hello World**. Все

эти строки можно найти в файле **res/values/strings.xml** и отредактировать при желании.

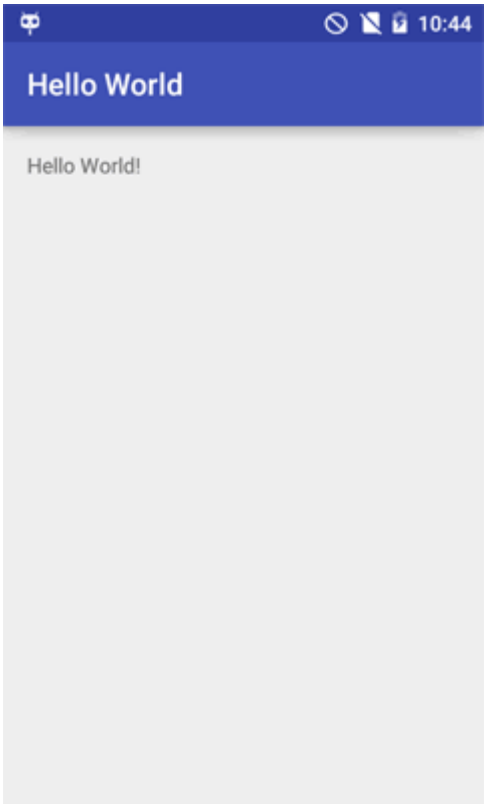

Рисунок П.2.2 – Запущенное приложение **Hello, World!**

Рассмотрим далее код приложения. Сначала изучим файл **activity_main.xml**.

Смотреть его можно в двух режимах - **Design** и **Text**.

Открываем в режиме **Text**.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
tools:context="ru.alexanderklimov.helloworld.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

</android.support.constraint.ConstraintLayout>
```

По коду видно, что имеется специальный контейнер **ConstraintLayout**, в котором размещён компонент **TextView**, предназначенный для вывода текста.

Далее расасмотрим Java-код (**MainActivity.java**).

```
package ru.username.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Это файл класса, где имя класса **MainActivity** совпадает с именем файла с расширением **java** (это правило, установленное

языком Java). В первой строке идет название пакета – оно было задано при создании проекта (Package Name). Далее идут строки импорта необходимых классов для проекта. Для экономии места они свёрнуты в одну группу. Разверните её. Если однажды имена классов выводятся серым цветом, значит, они не используются в проекте (подсказка **Unused import statement**) и эти строки можно удалить.

Далее идёт объявление самого класса, который наследуется (extends) от абстрактного класса **Activity**. Это базовый класс для всех экранов приложения.

В самом классе описан метод **onCreate()** – он вызывается, когда приложение создаёт и отображает разметку активности. Метод помечен как **protected** и сопровождается аннотацией **@Override** (переопределён из базового класса). Аннотация может пригодиться. Если вы сделаете опечатку в имени метода, то компилятор сможет предупредить об этом, сообщив об отсутствии такого метода у класса **Activity**.

Разберём код метода.

Строка **super.onCreate(savedInstanceState);** – это конструктор родительского класса, выполняющий необходимые операции для работы активности. Эту строчку не придётся трогать, оставляем без изменений.

Вторая строчка **setContentView(R.layout.activity_main);** представляет больший интерес. Метод **setContentView(int)** подключает содержимое из файла разметки. В качестве аргумента указываем имя файла без расширения из папки **res/layout**. По умолчанию проект создаёт в нём файл **activity_main.xml**. Можно переименовать файл или создать свой файл с именем **cat.xml** и подключить его к своей активности. Тогда код будет выглядеть так:

```
setContentView(R.layout.cat);
```

Чтобы код был аккуратным, старайтесь придерживаться стандартов. Если создается разметка для активности, то следует использовать префикс **activity_** для имени файла. Например,

разметка для второй активности может иметь имя **activity_second.xml**.

2.4 Модификация приложения

На данный момент созданное приложение слишком простое. Представьте себе, что у вас на экране должны располагаться несколько кнопок, текстовых полей, картинок. Каждому объекту нужно задать размеры, координаты, цвет, текст и так далее. Android поддерживает способ, основанный на XML-разметке, который будет напоминать разметку веб-страницы. Начинающие программисты могут использовать визуальный способ перетаскивания объектов с помощью мыши. Более продвинутые могут писать код вручную. Чаще используется комбинированный подход.

Файлы XML-разметки находятся в папке **res/layout** проекта. Слово "res" является сокращением от слова "resources" (ресурсы). Папка содержит ресурсы, не связанные с кодом. Кроме разметки, там же содержатся изображения, звуки, строки для локализации и т. д.

Раскройте слева в структуре проектов папки **res/layout** и дважды щелкните на файле **activity_main.xml**, если он у вас закрыт. Обратите внимание, что XML-файлы можно просматривать в двух режимах: текстовом и визуальном. Для этого предназначены две вкладки в нижней части окна редактора: **Design** и **Text** (рисунок П.2.3).

Переключаемся в режим **Text**.

Структура XML-файла достаточна проста – стандартное дерево XML-элементов, где каждый узел является именем класса **View** (**TextView** – один из элементов **View**). Можно создать интерфейс программы, используя структуру и синтаксис XML. Подобный подход позволяет разделить код программы и визуальное представление.

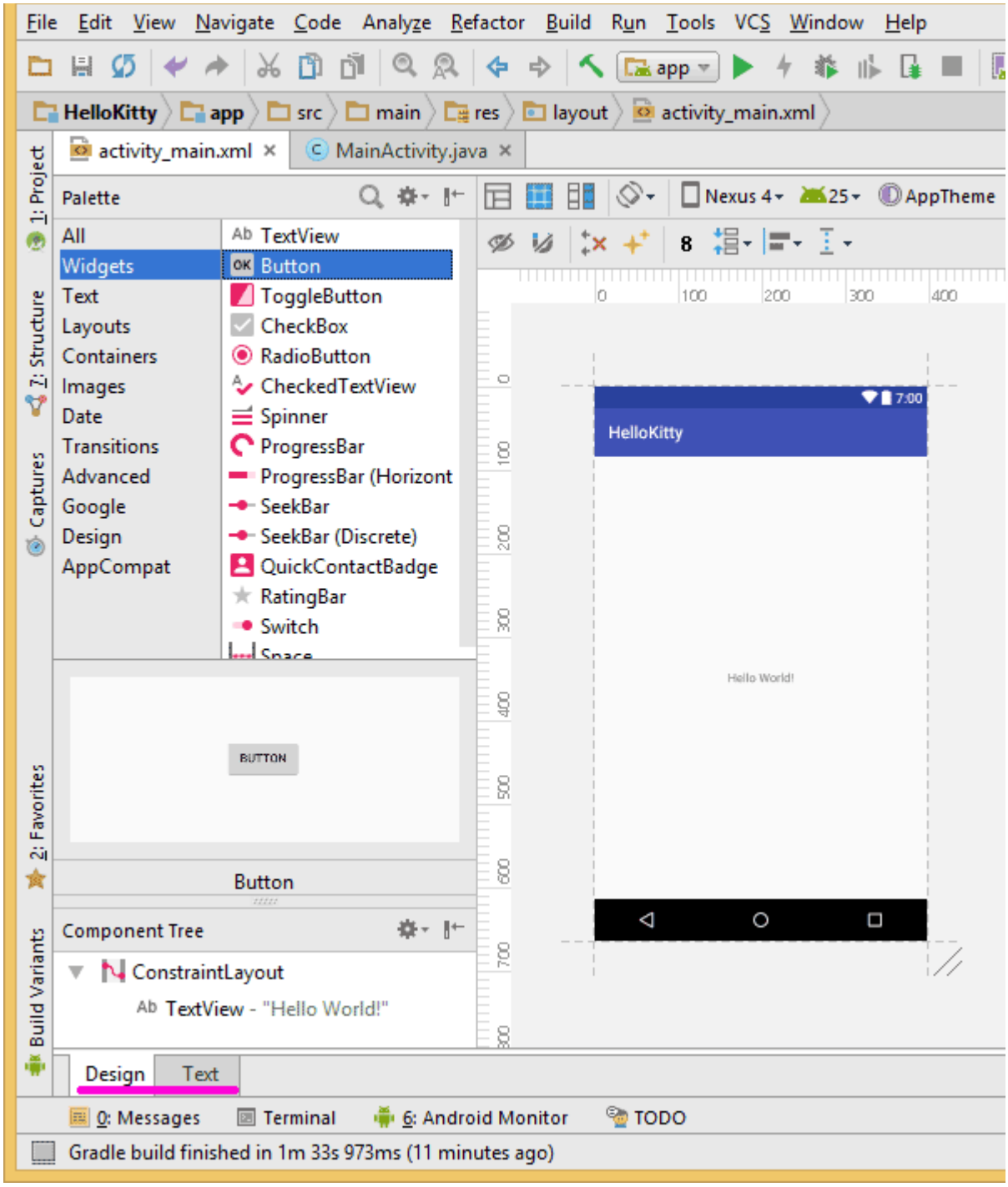

Рисунок П.2.3 – Окно редактора активности приложения

Можно продолжить работу над открытым проектом **Hello World** и модифицировать её под новые задачи либо создать новый проект, например, **Hello Kitty**.

Когда разметка открыта в графическом представлении, то слева от основной части редактора кода можно увидеть панель инструментов, в которой сгруппированы различные элементы по группам **Widgets**, **Texts**, **Layouts** и т. д. В группе **Images** находим элемент **ImageButton** и помещаем его на форму. Точное расположение в данном случае не имеет значения, просто стараемся разместить компонент в центре экрана активности. Появится диалоговое окно с просьбой выбрать изображение для кнопки. Пока можно выбрать любую.

Далее изменим фон для экрана приложения. Сейчас экран белого цвета. Возвращаемся в файл разметки **activity_main.xml**. Справа выбираем вкладку **Properties**, в которой отображаются свойства для выбранного элемента. А слева есть вкладка **Component Tree**, который отображает структуру компонентов на экране. Нужно выделить какой-нибудь компонент, что на вкладке свойств увидеть все доступные свойства компонента. Так как мы собираемся работать с фоном экрана приложения, то щёлкните на **ConstraintLayout**. В панели свойств отобразятся самые употребительные свойства выбранного компонента. К ним относятся идентификатор, ширина и высота (рисунок П.2.4).

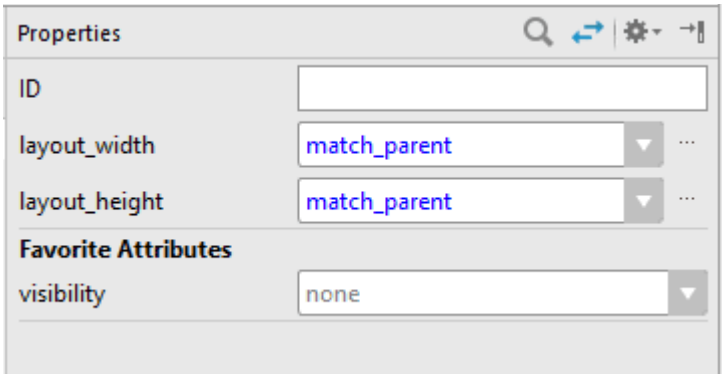

Рисунок П.2.4 – Панель свойств элементов управления

Щёлкаем по кнопке **View all properties** (две стрелочки), чтобы открыть все свойства компонента.

Находим свойство **background**. Щелкаем по кнопка с тремя точками, которая запустит диалоговое окно для создания ресурса (рисунок П.2.5).

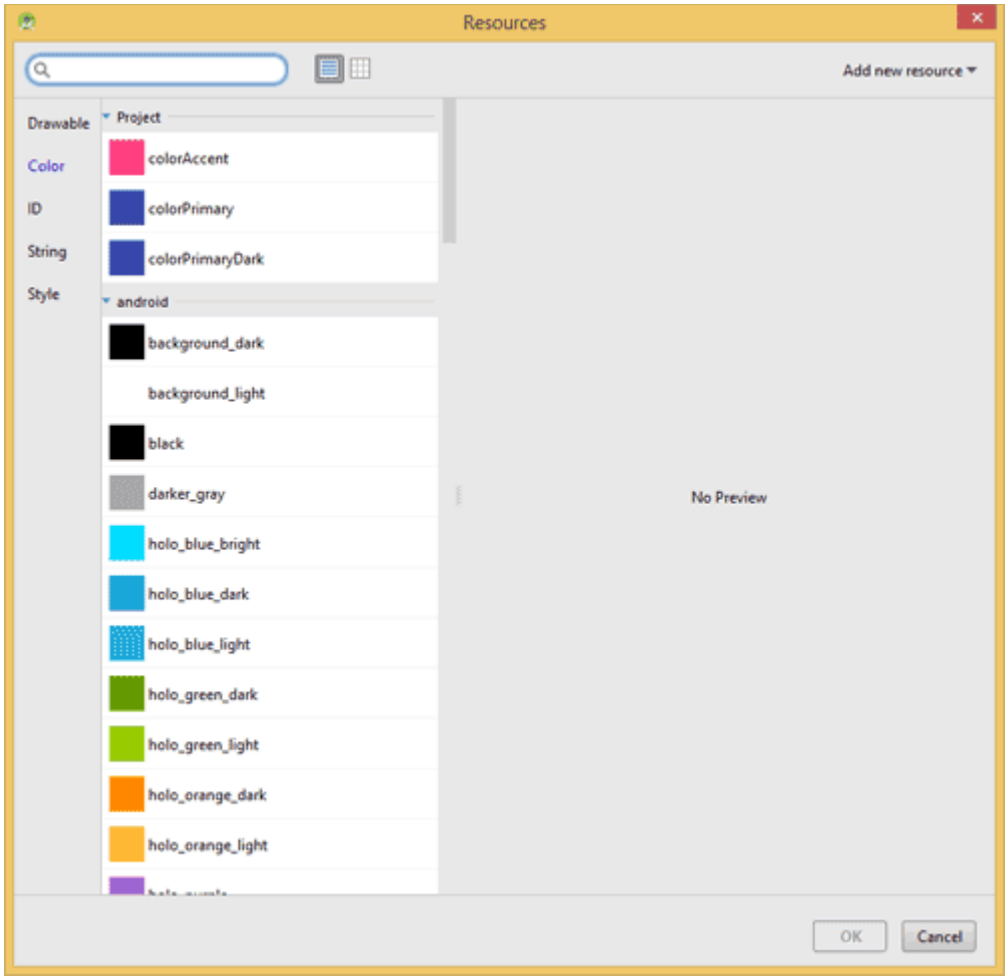

Рисунок П.2.5 – Диалоговое окно создания ресурса

Переходим на вкладку **Color** и выбираем цвет. Сверху в разделе **Project** показаны цвета, определённые в проекте в стиле Material Design. В разделе **Android** показаны цвета, существующие в ресурсах системы. При желании можно установить свой цвет. Выберем цвет **colorAccent** и нажмём кнопку **OK**.

Экран окрасится в розовый цвет. Если переключиться в текстовый режим, то видим, что у элемента **ConstraintLayout** добавилась строчка:

android:background="@color/colorAccent"

Таким образом, мы связали фон экрана с именем цветового ресурса. Существует неправильный подход, когда можно сразу напрямую указать значение цвета.

android:background="#ffffc0cb"

Но такого варианта следует избегать.

Далее поменяем картинку для графической кнопки. Находим любое понравившееся изображение и копируем его в папку **res/drawable**.

Простое перетаскивание из проводника в папку студии не сработает. Поэтому лучше скопировать картинку в буфер, затем щёлкнуть правой кнопкой мыши на папке **drawable** в студии, выбрать команду «Вставить» (рисунок П.2.6).

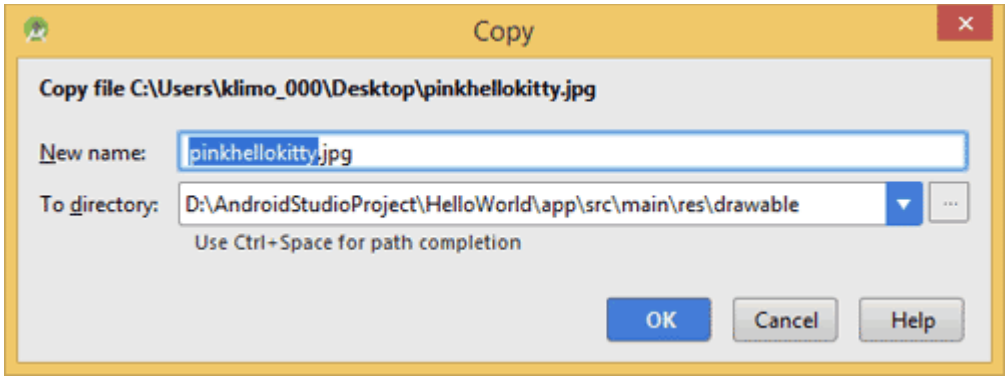

Рисунок П.2.6 – Копирование изображения в папку **res/drawable**

Когда вы поместите графический файл в указанную папку, то студия автоматически создаёт ресурс типа **Drawable** с именем файла без расширения, к которому можно обращаться программно.

Выделяем элемент **ImageButton** на форме и в панели свойств откроем только важные свойства, выбираем свойство **srcCompat**. Снова щёлкаем на кнопке с тремя точками и выбираем ресурс в категории **Drawable** – там должен появиться ресурс, имя которого совпадает с именем добавленного ранее файла (рисунок П.2.7).

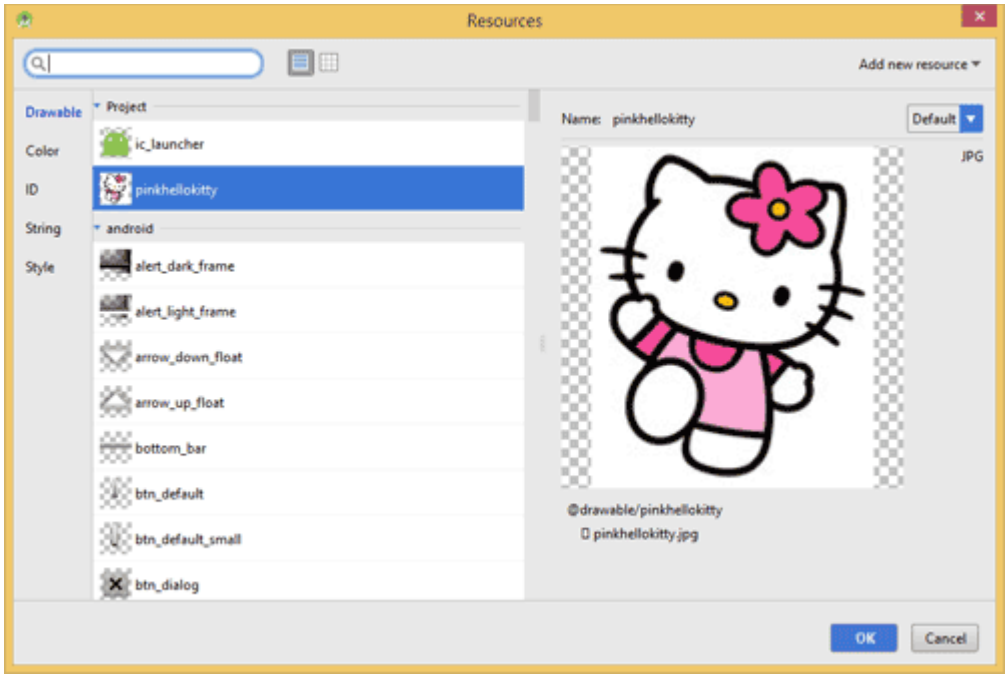

Рисунок П.2.7 – Выбор графического ресурса

Следует обратить внимание, что имена ресурсов должны начинаться с буквы и могут содержать буквы и цифры, а также знак нижнего подчеркивания. Другие символы типа тире, решётки и т. д. использовать нельзя.

Там же в окне свойств находим свойство **onClick** и вручную прописываем **onClick** – это будет именем метода для обработки нажатия на кнопку.

Мы закончили работу с графическим интерфейсом приложения. Напоследок, выделите элемент **TextView** с надписью **Hello, World** и в окне свойств посмотрите на его идентификатор (ID). В разделе **Widgets** найдите компонент **TextView** и перетащите его на форму приложения. Постарайтесь разместить его под графической кнопкой.

У этого компонента будет что-то написано в свойстве **id**. Скорее всего, это будет **textView**. Запомните его.

Если текст кажется мелковатым, то у свойства **textAppearance** можно установить значение **AppCompat. Display2**.

В результате получается примерно следующий код:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:background="@color/colorAccent"
   tools:context="ru.username.hellokitty.MainActivity">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello  World!"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"
      app:layout_constraintTop_toTopOf="parent"
      android:id="@+id/textView"
      android:layout_marginTop="0dp"
      app:layout_constraintVertical_bias="0.668"/>

   <ImageButton
      android:id="@+id/imageButton"
      android:layout_width="164dp"
      android:layout_height="199dp"
      android:layout_marginBottom="8dp"
      android:layout_marginLeft="8dp"
      android:layout_marginRight="8dp"
      android:layout_marginTop="8dp"
      android:onClick="onClick"
      android:scaleType="centerCrop"
      app:layout_constraintBottom_toBottomOf="parent"
```

```
app:layout_constraintHorizontal_bias="0.502"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.353"
app:srcCompat="@drawable/pinkhellokitty"
tools:layout_editor_absoluteX="104dp"
tools:layout_editor_absoluteY="101dp"/>

</android.support.constraint.ConstraintLayout>
```

Здесь много кода, который генерирует **ConstraintLayout** при размещении и выравнивании компонентов.

Установите курсор мыши внутри текста «**onClick**» у кнопки и нажмите комбинацию **Alt+Enter**

В всплывающем окне выберите вариант **Create 'onClick(View)' in 'MainActivity'** (рисунок П.2.8).

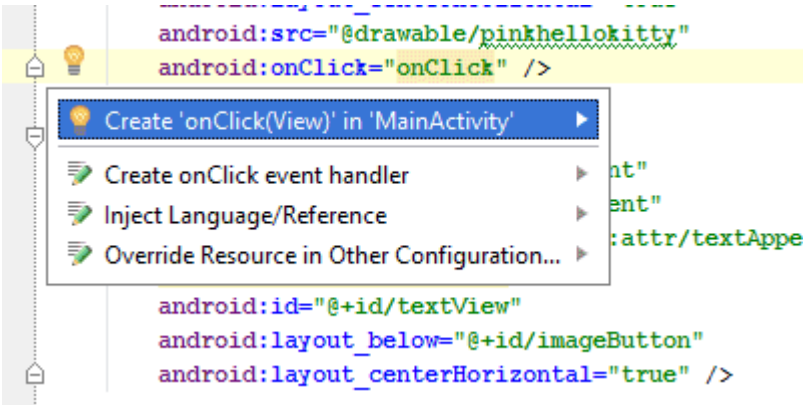

Рисунок П.2.8 – Создание обработчика события кнопки

В коде класса **MainActivity** появится заготовка для обработки щелчка кнопки.

Продолжаем работу в файле **MainActivity.java**. Так как мы собираемся менять текст в текстовой метке, необходимо прописать данный элемент в коде. До метода **onCreate()** наберите строчку:

private TextView mHelloTextView;

Мы объявили переменную типа **TextView** под именем **mHelloTextView**.

Если вы набирали вручную и при подсказках использовали клавишу Enter, то класс **TextView** автоматически импортируется и запись появится в секции **import**. Если вы просто копируете текст, то студия подчеркнёт название класса **TextView** и предложит импортировать его вручную.

Далее внутри метода **onCreate()** после вызова метода **setContentView()** добавьте строку:

mHelloTextView = (TextView)findViewById(R.id.textView);

Желательно код набирать самостоятельно и активно использовать автозавершение (Ctrl+Пробел) при наборе слов. Студия часто сама активно помогает подсказками. Теперь система знает о существовании элемента **TextView**, и мы можем к нему обращаться для изменения различных свойств, например, поменять текст.

Переходим к заготовке для щелчка кнопки.

public void onClick(View view) {

}

Пишем код между фигурными скобками:

mHelloTextView.setText("Hello Kitty!");

Мы обращаемся к элементу **mHelloTextView** и через его метод **setText()** программно меняем текст на нужные слова.

Запускаем программу и нажимаем на кнопку с изображением. Если всё сделано правильно, то отобразится фраза "Hello Kitty!".

В папке **app\build\outputs\apk** проекта можно найти готовый APK-файл, который можно выложить у себя на сайте и дать скачать кому-либо (в телефоне должно быть разрешение на установку неподписанных приложений).

Следует обратить внимание на то, что каждому приложению выделяется определённый объем памяти под картинки. То есть не нужно пытаться загрузить в этом примере фотографию

объёмом в несколько десятков мегабайт, иначе можно получить ошибку в приложении.

Продолжаем развивать приложение.

Найдите в разделе **Text Fields (EditText)** компонент **Plain Text** и перетащите его на экран активности, разместив где-то над картинкой. Оставляем все свойства без изменений, разве что в свойстве **hint** можно добавить строчку-подсказку, которая будет исчезать при вводе текста.

```
<EditText
   android:id="@+id/editText"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:ems="10"
   android:hint="Введите имя "
   android:inputType="textPersonName"
   android:layout_above="@+id/imageButton"
   android:layout_centerHorizontal="true" />
```

Переходим в класс **MainActivity** и добавляем новую переменную рядом с переменной **mHelloTextView**:

```
private EditText mNameEditText;
```

Свяжем созданную переменную с компонентом в методе **onCreate()**:

```
mNameEditText = (EditText) findViewById(R.id.editText);
```

Поменяем код для щелчка кнопки.

```
public void onClick(View view) {
   if (mNameEditText.getText().length() == 0) {
      mHelloTextView.setText("Hello Kitty!");
   } else {
      mHelloTextView.setText("Привет, " + mNameEdit-
Text.getText());
   }
}
```

Мы внесли небольшую проверку. Если в текстовом поле пустой текст, то длина текста составляет ноль символов, и мы по-прежнему выводим надпись "Hello Kitty!". Если пользователь введет другое имя, то приложение поздоровается с ним (рисунок П.2.9).

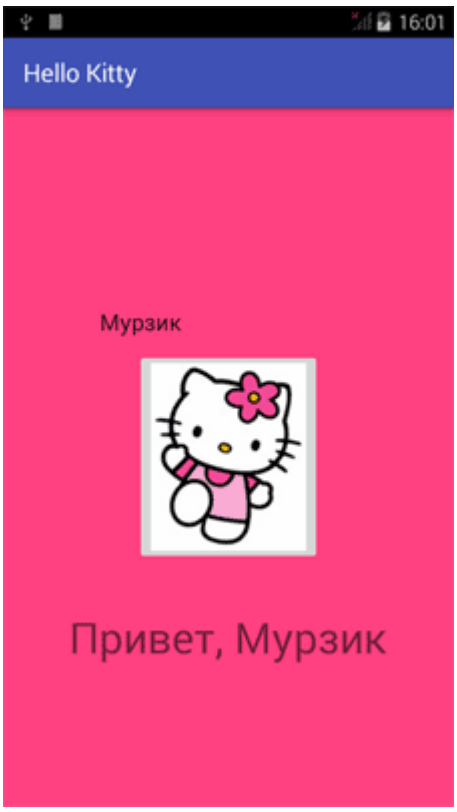

Рисунок П.2.9 – Результат работы приложения

3 Контрольные вопросы

1. Как создать простейшее приложение в Android Studio? Какие параметры при этом указываются?
2. Перечислите известные вам шаблоны активностей мобильных приложений.
3. Что содержится в папке манифеста приложения?
4. Какие виды ресурсов мобильных приложений вы знаете? В каких папках они хранятся?
5. Как менять свойства элементов управления?

6. Как создаются обработчики событий для элементов управления?

4 Задание для самостоятельного выполнения

1. Создайте новое мобильное приложение согласно макету рисунка П.2.10.

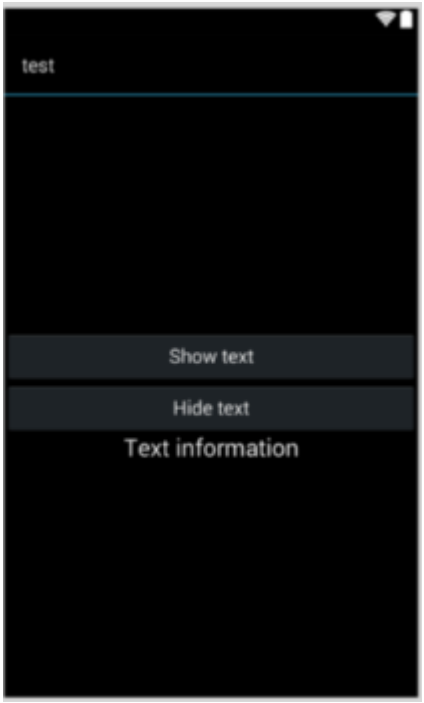

Рисунок П.2.10 – Макет приложения для задания 1

При нажатии на кнопку **“Show text”** отобразить текстовое поле **Text information**.

При нажатии на кнопку **“Hide text”** скрыть текстовое поле **Text information**.

2. Создайте новое мобильное приложение. Графический интерфейс окна должен выглядеть согласно макету рисунка П.2.11.

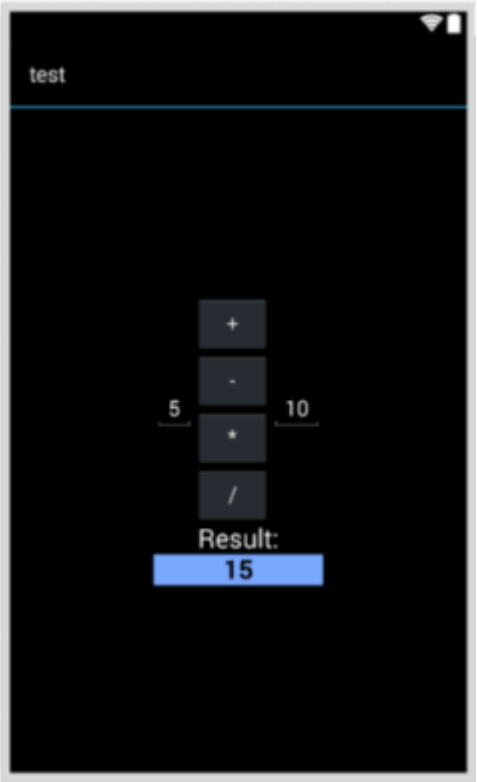

Рисунок П.2.11 – Макет приложения для задания 2

Программа должна выполнять функции калькулятора.

Построение графического интерфейса нужно произвести с помощью дизайнера окон и стандартных элементов управления (4 кнопки, 2 текстовых поля для ввода информации, 2 текстовых поля для вывода информации). Рекомендации по построению интерфейса:

– в качестве компоновщиков можно использовать **LinearLayout**;

– для выбора положения виджета в компоновщике используйте атрибут **gravity**;

– для установки высоты и ширины не забывайте о значениях **match_parent** и **wrap_content**;

– указывайте идентификаторы для элементов, которые Вам потом понадобятся в коде.

Рекомендации по использованию кода:

– определите в классе поля, которые будут хранить ссылки на кнопки и текстовые поля;

– в теле метода **OnCreate** получите доступ к виджетам, которые вы создали в дизайнере окон, с помощью метода **FindViewById**;

– определите обработчик события для каждой из 4-х кнопок. При этом используйте разные техники.

4. Разработать приложение согласно варианту:

1. В году примерно $3.156 \text{x} 10^7$сек. Написать программу, которая запрашивает возраст в годах и переводит его в секунды.

2. Масса *m* одной молекулы воды примерно равна $3 \text{x} 10^{-23}$ гр. Кварта воды равна примерно 950 гр. Написать программу, которая запрашивает количество воды в квартах и выводит число молекул в этом количестве воды.

3. Вычислить:

$$a = \frac{2\cos(x - 30^o)}{\frac{1}{3} + \sin^2(y+2)}, y = \frac{z^2}{5 + \frac{z^2}{7}}, x = 1.236, z = 4.5 .$$

4. Найти периметр и площадь прямоугольника.

5. Вычислить: $y = 2\ln x - tg(x)$.

6. Написать программу, которая запрашивает время в часах и минутах, после чего переводит в минуты, затем в секунды и выводит результаты на экран.

7. Найти площадь всей поверхности цилиндра: $T=2\pi R(R+H)$.

8. Вычислить: $y = x^5 + \cos(vt), x = \frac{v^3}{-t^2}, v = |t|, t - ввести$.

9. Найти диагональ и площадь квадрата.

10. Написать программу, которая запрашивает количество дней и переводит в недели и дни. Например, 18 дней = 2 недели и 4 дня.

11.Вычислить:

$$y = 2a - b \cdot c, a = \frac{\sqrt[4]{\frac{1}{2} - v \cdot |w^3 - 100|}}{b/13}, b = \frac{v}{w^2}, c = 1.5, w, v - ввести .$$

12.Найти площадь боковой поверхности шара: $T=4\pi R^2$.

13.Найти объем куба (с использованием стандартной функции и без).

14.Вычислить:

$$g = a + \left|\frac{\sin(q+a)}{b+c}\right|^{0.5}, При\ b = \frac{3.2a}{a+b+c}, c = 0.07a^2, a = 1.75, q = 48^o 24' 18''$$

15.Определить расстояние, пройденное физическим телом за время **t**, если тело движется с постоянной скоростью **v**.

16.Найти площадь поверхности куба.

17.Определить расстояние между двумя точками с координатами (x1,y1) и (x2,y2).

18.Вычислить: $y = \cos x - \frac{x^2 - 2x}{2 + (x^3 - 1)}$.

19.Найти объем цилиндра по формуле: $V=\pi R^2 H$.

20.Вычислить:

$$y = a + b + c, a = \frac{\sqrt[3]{w - 20}}{1 - (v + b/13)}, b = \frac{w}{v}, c = 25, w, v - ввести .$$

21.Найти расстояние от точки с координатами (x,y) до начала координат.

22.Вычислить:

$$y = \ln x + tg(z), x = \frac{2.4z}{1 - z^2}, z = \sqrt{|2v - 10|}, v = 20.3 .$$

23.Даны длины сторон оснований усеченной пирамиды и высота. Найти объем пирамиды по формуле: $V = \frac{H}{3}(S_1 + S_2 + \sqrt{S_1 S_2})$, где S_1, S_2 – площади оснований усеченной пирамиды, H – высота.

24.Заданы уравнения двух пересекающихся прямых на

плоскости: $y_1=k_1+b_1$; $y_2=k_2+b_2$. Найти (в градусах и минутах) угол между ними, используя формулу: $\mathrm{tg}\varphi = \dfrac{(k_2 - k_1)}{(1 + k_1 k_2)}$.

25.Русские неметрические единицы длины: 1 верста=500 саженей; 1 сажень=3 аршина; 1 аршин=16 вершков; 1 вершок=44,45 мм. Длина некоторого отрезка составляет *p* метров. Перевести ее в русскую неметрическую систему.

26.Дано четырехзначное число. Найти произведение первой и третьей цифры, просуммировать полученное произведение с последней цифрой.

27.Трехмерные вектора заданы своими координатами: $A=(x_a, y_a, z_a)$ и $B=(x_b, y_b, z_b)$. Найти угол (в градусах) между векторами A и B, используя формулу: $\cos\varphi = \dfrac{(A,B)}{|A| \cdot |B|} = \dfrac{x_a x_b + y_a y_b + z_a z_b}{\sqrt{x_a^2 + y_a^2 + z_a^2} \cdot \sqrt{x_b^2 + y_b^2 + z_b^2}}$.

28.В равнобедренном прямоугольном треугольнике известна высота *h*, опущенная на гипотенузу. Найти стороны треугольника.

29.Угол α задан в радианах. Найти его величину в градусах, минутах и секундах.

30.Дано трехзначное число. Найти сумму его цифр.

31.Заданы моменты начала (t_1) и конца (t_2) некоторого промежутка времени в часах, минутах и секундах (в пределах одних суток). Найти продолжительность этого промежутка в тех же единицах измерения.

32.Найти координаты вершины параболы $y = ax^2 + bx + c$.

Практическая работа №3. Работа со всплывающими сообщениями

1 Цель работы

Получить практические навыки создания всплывающих сообщений при разработке мобильных приложений.

2 Порядок выполнения работы

2.1 Общие сведения о всплывающих сообщениях

Всплывающее уведомление (Toast Notification) является сообщением, которое появляется на поверхности окна приложения, заполняя необходимое ему количество пространства, требуемого для сообщения. При этом текущая деятельность приложения остается работоспособной для пользователя. В течение нескольких секунд сообщение плавно закрывается. Всплывающее уведомление также может быть создано службой, работающей в фоновом режиме. Как правило, всплывающее уведомление используется для показа коротких текстовых сообщений.

Для создания всплывающего уведомления необходимо инициализировать объект **Toast** при помощи метода **Toast.makeText()**, а затем вызвать метод **show()**для отображения сообщения на экране:

```
Toast toast = Toast.makeText(getApplicationContext(),
  "Пора покормить кота!", Toast.LENGTH_SHORT);
toast.show();
```

У метода **makeText()** есть три параметра:

- контекст приложения;
- текстовое сообщение;
- продолжительность времени показа уведомления. Можно использовать только две константы: **LENGTH_SHORT** (по умолчанию) – показывает текстовое уведомление на короткий

промежуток времени, **LENGTH_LONG** – показывает текстовое уведомление в течение длительного периода времени.

В исходниках Android можно найти следующие строчки:

```
private static final int LONG_DELAY = 3500; // 3.5 seconds
private static final int SHORT_DELAY = 2000; // 2 seconds
```

Как видно, уведомления выводятся на 3 с половиной секунды или на 2 секунды. Других вариантов не имеется.

По умолчанию стандартное всплывающее уведомление появляется в нижней части экрана. Изменить место появления уведомления можно с помощью метода **setGravity(int, int, int)**. Метод принимает три параметра:

- стандартная константа для размещения объекта в пределах большего контейнера (например, **GRAVITY.CENTER**, **GRAVITY.TOP** и др.);
- смещение по оси X;
- смещение по оси Y.

Например, если надо, чтобы уведомление появилось в центре экрана, то следует использовать следующий код:

```
toast.setGravity(Gravity.CENTER, 0, 0);
```

Если нужно сместить уведомление направо, то просто увеличьте значение второго параметра. Для смещения вниз нужно увеличить значение последнего параметра. Соответственно, для смещения вверх и влево используйте отрицательные значения.

Типичная ошибка начинающих программистов – забывают добавить вызов метода **show()** для отображения сообщения на экране. К счастью, в студии, если вы пропустите метод **show()**, то строка будет подсвечена, а при подведении указателя мыши к строке появится подсказка: «Toast created but not shown: did you forget to call show()?»

2.2 Пример создания всплывающих сообщений

Создайте новый проект или используйте любой старый проект из предыдущих занятий. Добавьте на экран активности

кнопку с текстом **Показать Toast**, а также присвойте атрибуту **android:onClick** значение **showToast**. Далее напишем код:

```
public void showToast(View view) {
    //создаем и отображаем текстовое уведомление
    Toast toast = Toast.makeText(getApplicationContext(),
            "Пора покормить кота!",
            Toast.LENGTH_SHORT);
    toast.setGravity(Gravity.CENTER, 0, 0);
    toast.show();
}
```

Запустите приложение и нажмите кнопку. В центре экрана появится на короткое время текстовое сообщение, которое само исчезнет (рисунок П.3.1).

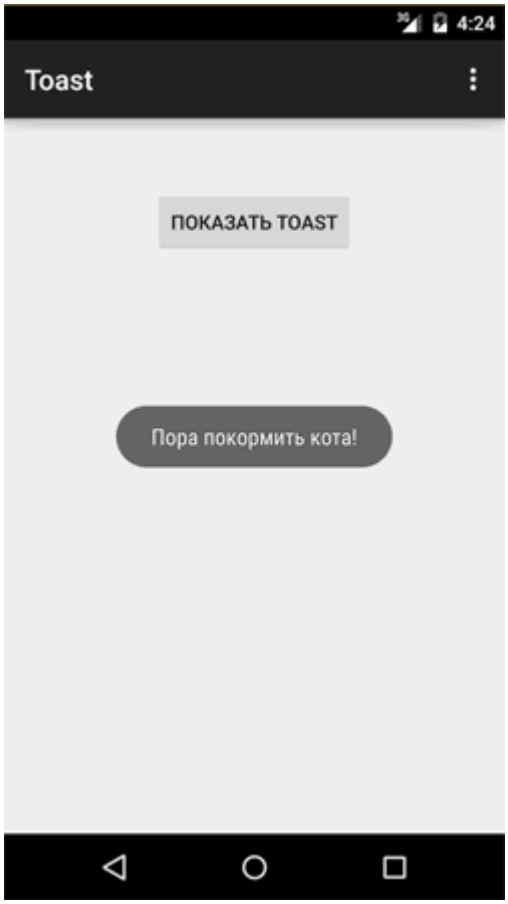

Рисунок П.3.1 – Внешний вид работающего приложения

Разберем еще один пример. Удалим предыдущий код для щелчка кнопки и напишем следующее:

```
int duration = Toast.LENGTH_LONG;
Toast toast2 = Toast.makeText(getApplicationContext(),
        R.string.catfood,
        duration);
toast2.setGravity(Gravity.TOP,  0, 0);
toast2.show();
```

На этот раз используем константу **LENGTH_LONG**. Помимо этого, текст сообщения помещаем в XML-ресурсы. Кроме того, сообщение будет выводиться в верхней части экрана.

2.3 Добавление рисунка в уведомление

Как правило, для **Toast** используются короткие текстовые сообщения. При необходимости можно добавить к сообщению и картинку. Используя метод**setView()**, принадлежащий объекту **Toast**, можно задать любое представление (включая разметку) для отображения.

Подготовьте картинку и разместите её в папке **res/drawable**, как было сделано в предыдущей работе. Картинка будет доступна приложению как ресурс через название файла без расширения. Чтобы изображение появилось в стандартном Toast-сообщении, потребуется программно создать объект класса **ImageView** и задать для него изображение из ресурсов с помощью метода **setImageResource**. Сам по себе стандартный внешний вид **Toast** состоит из контейнера **LinearLayout**, в который нужно добавить созданный объект **ImageView**. Можно задать также позицию, в которую следует вывести изображение. Если указать значение 0, то изображение будет показано выше текста. Код для создания **Toast** с изображением выглядит следующим образом, а внешний вид запущенного приложения показан на рисунке П.3.2:

```
public void showToast(View view) {
    Toast toast3 = Toast.makeText(getApplicationContext(),
            R.string.catfood, Toast.LENGTH_LONG);
    toast3.setGravity(Gravity.CENTER, 0, 0);
    LinearLayout toastContainer = (LinearLayout)
            toast3.getView();
    ImageView catImageView = new ImageView (getApplica-
            tionContext());
    catImageView.setImageResource(R.drawable.hungrycat);
    toastContainer.addView(catImageView, 0);
    toast3.show();
}
```

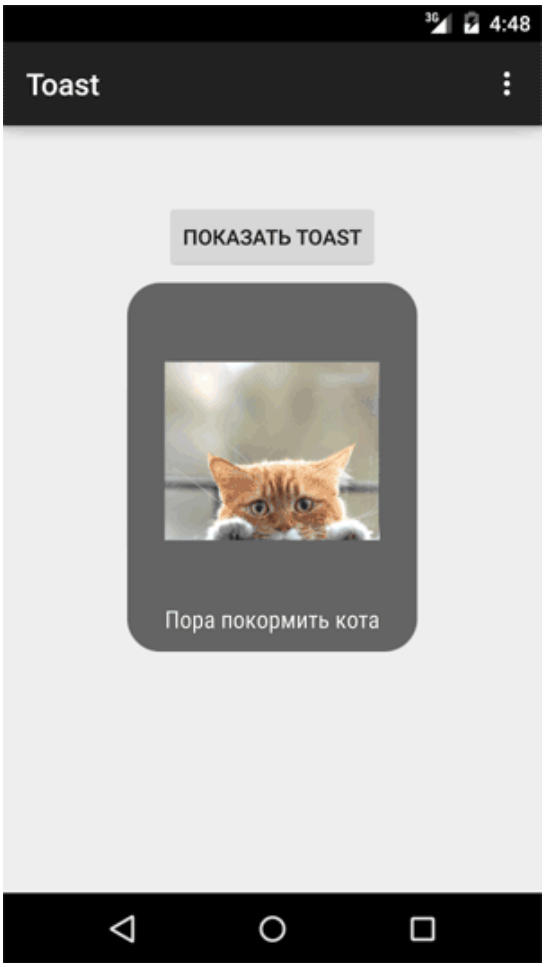

Рисунок П.3.2 – Уведомление с рисунком

2.4 Создание собственных всплывающих уведомлений

В предыдущем примере мы получили доступ к контейнеру через метод **getView()**. Можно пойти от обратного – подготовить свой контейнер и внедрить его в объект **Toast** через метод **setView()**.

Если простого текстового сообщения недостаточно для уведомления пользователя приложения, можно создать собственный дизайн разметки своего уведомления. Для получения разметки из XML-файла и работы с ней в программе используется класс **LayoutInflater** и его методы **getLayoutInflater()** или **getSystemService()**, которые возвращают объект **LayoutInflater**. Затем вызовом метода **inflate()** получают корневой объект **view** этой разметки. Например, для файла разметки уведомления с именем **custom_layout.xml** и его корневого элемента с идентификатором **android:id="@+id/toast_layout"** код будет таким:

```
LayoutInflater  inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_layout,
            (ViewGroup) findViewById(R.id.toast_layout));
```

Параметры, передаваемые в метод **inflate()**:

- идентификатор ресурса схемы размещения (custom_layout.xml);
- идентификатор ресурса корневого представления (toast_layout).

После получения корневого представления из него можно получить все дочерние представления уже известным методом **findViewByid()** и определить информационное наполнение для этих элементов.

Затем создается объект **Toast** и устанавливаются нужные свойства, например, **Gravity** и продолжительность времени показа уведомления.

```
Toast toast = new Toast(getApplicationContext());
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
toast.setDuration(Toast.LENGTH_LONG);
```

После этого вызывается метод **setView()**, которому передаётся разметка уведомления, и метод **show()**, чтобы отобразить уведомление с собственной разметкой:

```
toast.setView(layout);
toast.show();
```

При этом нужно создать разметку **custom_layout.xml**.

Определите два дочерних элемента **ImageView** и **TextView**:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/toast_layout"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="10dp"
    android:background="#DAAA">
<ImageView android:id="@+id/imageView"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_marginRight="10dp"/>
<TextView android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:text="@string/catfood"
    android:textColor="#FFF777">
</LinearLayout>
```

Запустите проект на выполнение. При нажатии кнопки вызова должно появиться на несколько секунд окно уведомления с текстовым сообщением и значком.

2.5 Использование уведомлений Toast в рабочих потоках

Как элемент графического интерфейса **Toast** должен быть вызван в потоке GUI, иначе существует риск выброса межпотокового исключения. В листинге объект **Handler** используется для гарантии того, что уведомление **Toast** было вызвано в потоке GUI.

```
private void mainProcessing() {
   Thread thread = new Thread(null, doBackgroundThread-
Processing,
         "Background");
    thread.start();
}
private Runnable doBackgroundThreadProcessing = new Run-
nable() {
   public void run() {
      backgroundThreadProcessing();
   }
};
private void backgroundThreadProcessing() {
   handler.post(doUpdateGUI);
}
// Объект Runnable, который вызывает метод из потока GUI
private Runnable doUpdateGUI = new Runnable() {
   public void run() {
      Context context = getApplicationContext();
      String msg = "To open mobile development!";
      int duration = Toast.LENGTH_SHORT;
      Toast.makeText(context, msg, duration).show();
   }
};
```

Напоследок следует предупредить об одной потенциальной проблеме. При вызове сообщения нужно указывать контекст в

первом параметре метода **makeText()**. В интернете попадаются примеры типа **makeText(MainActivity.this, ...)**. Ошибки в этом нет, так как класс **Activity** является потомком **Context** и в большинстве случаев пример будет работать. Но иногда могут возникнуть ситуации, когда текст не выравнивается, обрезается и т. д. Это связано с тем, что активность может использовать определённую тему или стиль, которые вызывают такой побочный эффект. Поэтому рекомендуется использовать метод **getApplicationContext()**.

Второй момент – фрагменты не являются потомками контекста. Если вы захотите вызвать всплывающее сообщение во фрагменте, то возникнет проблема. В этом случае нужно добавить новую переменную класса **Activity** через метод **getActivity()**:

```
Activity activity = getActivity();
Toast.makeText(activity, "Кота покормили?", Toast.
                    LENGTH_SHORT).show();
```

То же самое может случиться при вызове всплывающих сообщений из диалоговых окон, которые тоже не относятся к классу **Context**. Вместо **getApplicationContext()** также можно вызывать метод **getBaseContext()**.

3 Контрольные вопросы

1. Что такое всплывающее сообщение? Какой класс отвечает за его создание?
2. Какова продолжительность времени показа уведомления?
3. Как установить место показа всплывающего сообщения на экране?
4. Как добавить в уведомление рисунок?
5. Как создать собственный дизайн разметки уведомления?

4 Задание для самостоятельного выполнения

Добавьте в приложение «Калькулятор» практической работы №2 возможность вывода всплывающего сообщения в случае попытки деления на ноль. Сообщение должно содержать соответствующий текст и произвольную картинку.

Практическая работа №4. Работа с меню

1 Цель работы

Получить практические навыки создания и модификации меню при разработке мобильных приложений.

2 Порядок выполнения работы

2.1 Создание меню в мобильных приложениях

Операционная система Android поддерживает несколько типов меню. Первый – на телефоне есть отдельная кнопка **Menu** (в старых телефонах), нажатие которой вызывает меню. В новых устройствах отдельную кнопку убрали, заменив на значок меню в виде трёх точек в вертикальной ориентации. Второй тип – контекстное меню, которое появляется при нажатии и удерживания пальца на экране в нужном месте (также можно нажать и удерживать центральную кнопку на телефоне). Контекстное меню в свою очередь может иметь подменю. В данной работе мы будем использовать первый тип меню для новых устройств под управлением Android 4.0 и выше.

В шаблоне **Empty Activity** нет меню, поэтому мы создадим его сами. Это поможет понять принцип работы и получить общее представление о проекте. Запоминать названия классов, методов и код для обработки выбора пунктов меню необязательно. В других шаблонах меню будет встроено и можно будет сразу его использовать.

Создайте новый проект на основе **Empty Activity** и запустите его. Никакого меню пока нет.

Создадим несколько строковых ресурсов в файле **res/values/strings.xml**, которые будут отвечать за пункты меню:

```
<string name="action_settings">Settings</string>
<string name="action_cat_male">Кот</string>
```

```
<string name="action_cat_female">Кошка</string>
<string name="action_kitten">Котёнок</string>
```

Теперь создайте новую папку **menu** в папке **res** (правый щелчок мыши на папке **res**, | **New** | **Directory**). Далее создайте в созданной папке файл **menu_main.xml.** Имя указывает, что меню относится к основной активности **MainActivity** (правый щелчок мыши на папке **menu** | **New** | **Menu Resource File**). Если вы будете создавать приложение с несколькими экранами, то у каждой активности будет отдельное меню со своими настройками. Откроем файл **menu_main.xml** и добавим в полученный шаблон свой код:

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      tools:context=".MainActivity">
    <item
      android:id="@+id/action_settings"
      android:orderInCategory="100"
      android:title="@string/action_settings"
      app:showAsAction="never"/>
</menu>
```

Откроем файл **MainActivity**. Сейчас в нём только один метод **onCreate()**. Добавим новый метод **onCreateOptionsMenu()**. Именно данный метод отвечает за появление меню у активности. Выберите в студии меню **Code** | **Override Methods...** и в следующем окне начинайте вводить название метода по первым буквам (рисунок П.4.1). Можно вводить по первым заглавным буквам, т.е. **OCOM** (**o**n**C**reate**O**ptions**M**enu), чтобы быстро найти нужную строку. Нажимаем кнопку **OK** и получаем заготовку.

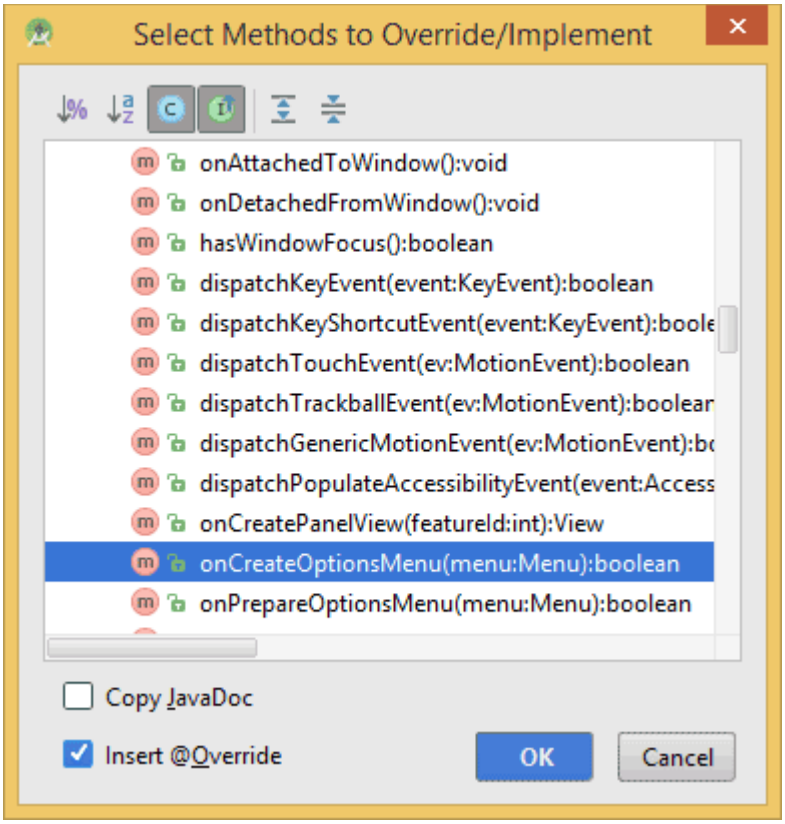

Рисунок П.4.1 – Создание переопределенного метода

```
@Override
public boolean onCreateOptionsMenu(Menu  menu) {
   return super.onCreateOptionsMenu(menu);
}
```

Добавляем в заготовку метод, который берёт данные из ресурсов меню и преобразует их в пункты меню на экране.

```
@Override
public boolean onCreateOptionsMenu(Menu  menu) {
   getMenuInflater().inflate(R.menu.menu_main,  menu);
   return true;
}
```

В методе **inflate()** вы указываете на ресурс меню (**R.menu. menu_main**) и объект класса **Menu**.

Запустите проект. В правой части заголовка активности появится значок из трёх точек, выстроенных в вертикальную линию (рисунок П.4.2). Нажмите на значок, чтобы увидеть пункт меню **Settings**.

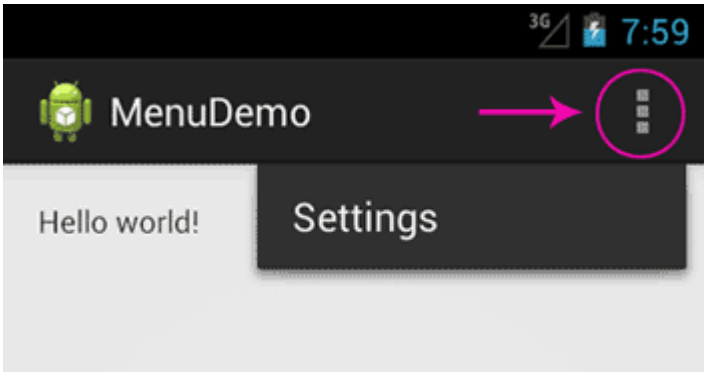

Рисунок П.4.2 – Приложение с созданным меню

Как не трудно догадаться, элемент **item** отвечает за отдельный пункт меню. Добавим ещё три пункта по такому же принципу, меняя только идентификатор и текст для меню:

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        tools:context=".MainActivity">
      <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never"/>

      <item
        android:id="@+id/action_cat1"
        android:orderInCategory="100"
```

```
        android:title="@string/action_cat_male"
        app:showAsAction="never"/>

    <item
        android:id="@+id/action_cat2"
        android:orderInCategory="100"
        android:title="@string/action_cat_female"
        app:showAsAction="never"/>

    <item
        android:id="@+id/action_cat3"
        android:orderInCategory="100"
        android:title="@string/action_kitten"
        app:showAsAction="never"/>
</menu>
```

Запустите проект и попробуйте снова вызвать меню. Вы увидите три новых пункта (рисунок П.4.3).

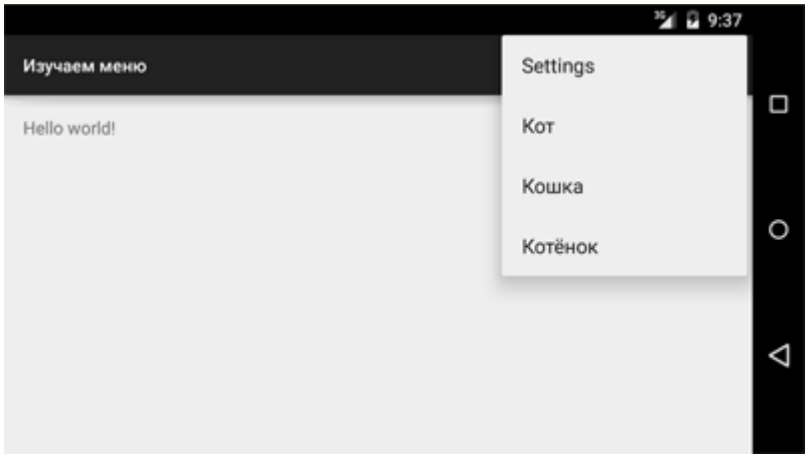

Рисунок П.4.3 – Добавление пунктов меню

Параметры **id** и **title** не нуждаются в объяснениях. Параметр **orderInCategory** позволяет задать свой порядок вывода пунктов меню. Предположим, вы создали пять пунктов меню, но пока не определились с порядком их вывода на экране. Что-

бы не перемещать постоянно целые блоки кода для пунктов меню в нужном порядке, можно воспользоваться данным параметром.

И, наконец, важный атрибут **app:showAsAction** определяет поведение меню в **ActionBar**. Значение **never** означает, что элемент меню не должен выводиться в заголовке, а только во всплывающем меню, т.е. находиться за тремя точками. Если вы установите значение **always**, то пункт **Settings** сразу появится в заголовке приложения. Также доступны значения **ifRooms**, **withText** и **collapseActionView**. Например, **ifRoom** выводит пункт меню, если позволяет место. Если пунктов будет много, то они будут отвлекать внимание. Как правило, в таком варианте выводят очень короткое слово или значок для частых операций, чтобы избежать лишнего щелчка на три точки.

Обратите внимание на атрибут **app:showAsAction**, который относится к пространству имён **xmlns:app="http://schemas.android.com/apk/res-auto"**. Прежде такого пространства имён не существовало, и в проектах использовался атрибут **android:showAsAction** из стандартного пространства имён. Отредактируйте код в случае ошибки.

Пока пункты меню не выполняют полезной работы. Любое нажатие на пункт просто закрывает меню без видимых последствий.

2.2 Выбор пунктов меню

На данном этапе меню бесполезно, так как его пункты никак не реагируют на нажатия. Для обработки нажатий пунктов меню служит другой метод – **onOptionsItemSelected()**. Добавим метод по такому же принципу, как для предыдущего примера. Получим заготовку.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   return super.onOptionsItemSelected(item);
}
```

Параметр **item** отвечает за пункт меню. Следует получить идентификатор меню через метод **getItemId()** и указать для него код. Так как обычно меню состоит из нескольких пунктов, то удобно использовать конструкции **if/else** или **switch**. Для вывода информации воспользуемся текстовой меткой. Добавьте на экран активности компонент **TextView**. Можно использовать имеющийся **TextView** с надписью "Hello World!", только необходимо присвоить ему идентификатор.

```
<TextView
   android:id="@+id/textView"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Hello  World!"/>
```

Добавим код в заготовку для выбранного пункта меню:

```
@Override
public boolean onOptionsItemSelected(MenuItem  item) {
   // получим идентификатор выбранного пункта меню
   int id = item.getItemId();
   TextView  infoTextView  = (TextView)  findViewById
               (R.id.textView);
   // Операции  для выбранного пункта  меню
   switch (id) {
      case R.id.action_cat1:
         infoTextView.setText("Вы  выбрали кота!");
         return true;
      case R.id.action_cat2:
         infoTextView.setText("Вы  выбрали кошку!");
         return true;
      case R.id.action_cat3:
         infoTextView.setText("Вы  выбрали котёнка!");
         return true;
      default:
```

```
        return super.onOptionsItemSelected(item);
    }
}
```

Запустите приложение, вызовите меню и выберите любой пункт меню. В текстовом поле должно появиться сообщение.

Существует альтернативный способ обработки нажатий пунктов меню через XML, похожий на обработку щелчков кнопки. Вы можете добавить атрибут **android:onClick** в ресурсах меню, тогда не нужно использовать вызов метода **onOptionsItemSelected()**. При помощи **android:onClick** указывается нужный метод при выборе пункта меню. Добавьте данный атрибут к пункту **Settings**

```
<item
    android:id="@+id/action_settings"
    android:title="@string/action_settings"
    android:orderInCategory="100"
    android:onClick="onSettingsMenuClick"
    app:showAsAction="never" />
```

Теперь в коде активности напишем следующее:

```
// у атрибута пункта меню Settings установлено значение
// android:onClick="onSettingsMenuClick"
public void onSettingsMenuClick(MenuItem item) {
    TextView infoTextView = (TextView) findViewById(R.id.textView);
    infoTextView.setText("Вы выбрали пункт Settings, лучше бы выбрали кота");
}
```

Внешний вид пунктов меню можно изменить на переключатели. Для этого нужно добавить элемент **group** с атрибутом **android:checkableBehavior="single"**:

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity">
    <item
      android:id="@+id/action_settings"
      android:orderInCategory="100"
      android:title="@string/action_settings"
      app:showAsAction="never" />

    <group android:checkableBehavior="single">

      <item
        android:id="@+id/action_cat1"
        android:orderInCategory="100"
        android:title="@string/action_cat_male"
        app:showAsAction="never" />

      <item
        android:id="@+id/action_cat2"
        android:orderInCategory="100"
        android:title="@string/action_cat_female"
        app:showAsAction="never" />

      <item
        android:id="@+id/action_cat3"
        android:orderInCategory="100"
        android:title="@string/action_kitten"
        app:showAsAction="never" />
    </group>
</menu>
```

Начиная с Android Studio 2.2 был добавлен графический режим построения меню, которых похож на панель инструмен-

тов для добавления новых компонентов на экран (рисунок П.4.4). Панель состоит из четырёх элементов: **Menu Item**, **Search Item**, **Menu**, **Group**.

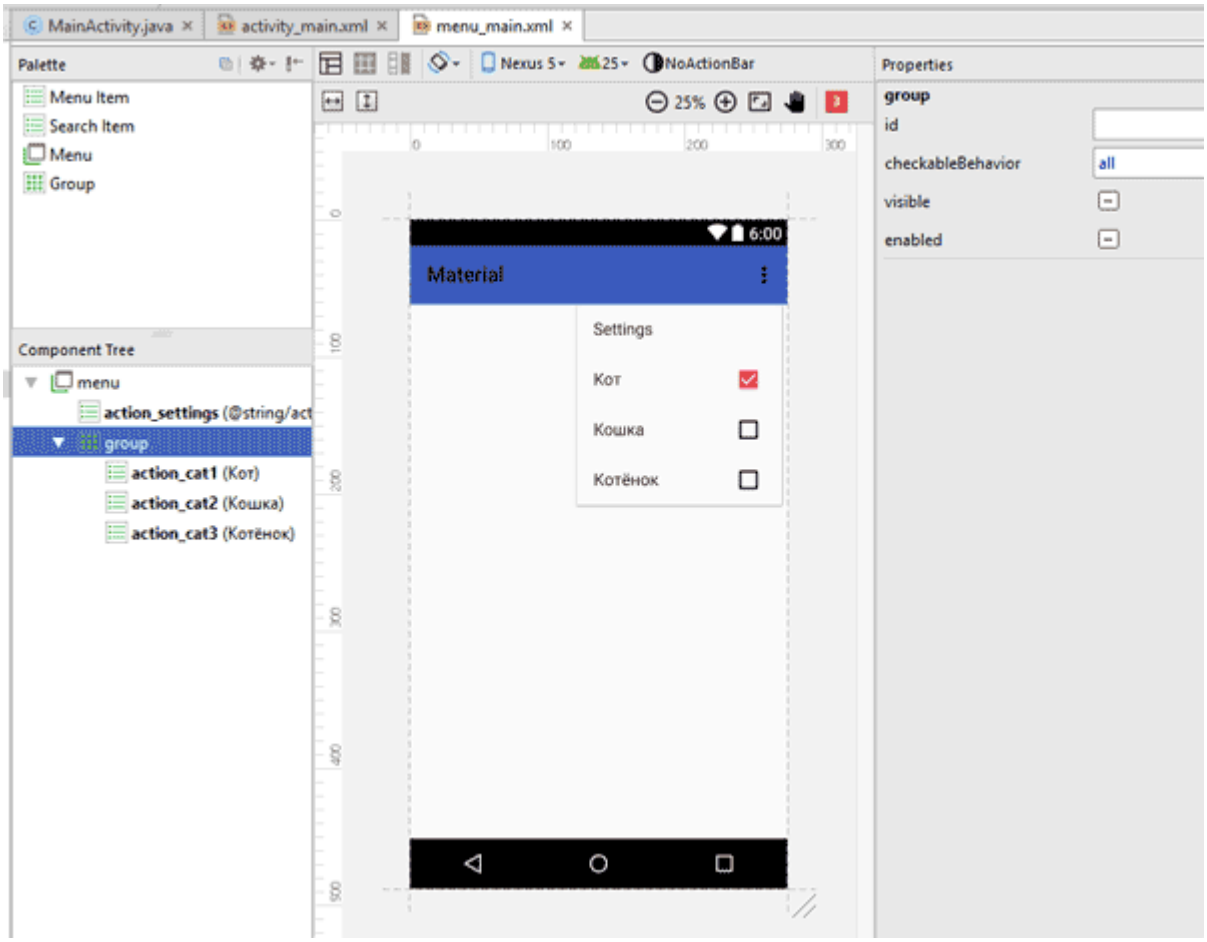

Рисунок П.4.4 – Графический режим создания меню

Принцип в данном случае аналогичен, выбирается нужный элемент и перетаскивается на экран в область меню. С его помощью можно быстро разработать структуру меню, а затем подправить вручную.

3 Контрольные вопросы

1. Опишите типы меню Android.
2. Как создать меню для проекта типа EmptyActivity?

3. Какие параметры позволяют задать внешний вид и поведение пунктов меню?

4. Как создать обработчик нажатия пункта меню?

5. Как изменить внешний вид пункта меню на переключатель?

4 Задание для самостоятельного выполнения

Модифицируйте приложение «Калькулятор» практической работы №2 путем добавления в него меню. Элементы меню должны дублировать кнопки «+», «–», «*», «/».

Лабораторная работа №1. Работа с ресурсами

1 Цель работы

Цель работы – изучить определение и варианты использования строковых и графических ресурсов при разработке мобильных приложений в Android Studio.

2 Содержание отчета

- титульный лист;
- краткое теоретическое описание;
- задание на лабораторную работу, включающее математическую формулировку задачи;
- результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений.

3 Порядок выполнения работы

В качестве примера напишем приложение под условным названием «Светофор». Интерфейс программы будет выглядеть следующим образом. На красном экране расположены три кнопки и одна текстовая надпись. При нажатии кнопок фон программы будет меняться на соответствующий цвет, который закреплён за определённой кнопкой. При этом покажем решение задачи с разных сторон.

Создайте новый проект на основе «Hello, World» и перетащите с панели инструментов две кнопки.

В окне **Component Tree** выделите строку **button**. В результате появится окно свойств **Properties**. Для первой кнопки присвоим свойству **id** значение **buttonRed** вместо стандартного **@+id/button.** Для второй кнопки присвоим значение **buttonYellow** (рисунок Л.1.1).

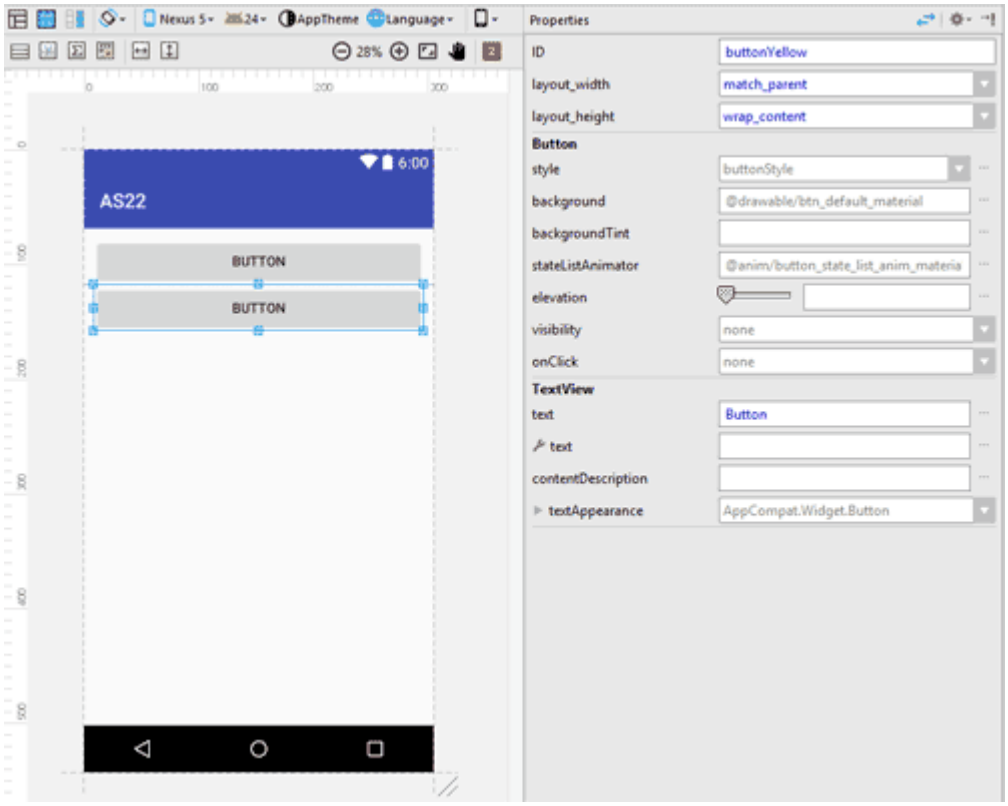

Рисунок Л.1.1 – Изменение идентификаторов кнопок

Третью кнопку создадим не через визуальное проектирование, а через код. Для этого в главном окне переключитесь с вкладки **Design** на вкладку **Text**. Здесь находится XML-разметка программы, в том числе и код для двух кнопок.

Для создания третьей кнопки нужно просто взять за образец код любой из двух кнопок и добавить под ними новую строчку перед закрывающим тегом </RelativeLayout> (не забудьте изменить идентификатор и атрибут android:layout_below):

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:text="@string/hello_world" />

<Button
    android:id="@+id/buttonRed"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:text="Button" />

<Button
    android:id="@+id/buttonYellow"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/buttonRed"
    android:text="Button" />

<Button
    android:id="@+id/buttonGreen"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/buttonYellow"
    android:text="Button" />

</RelativeLayout>
```

Настоятельно рекомендуется не копировать строку, а написать ее вручную. Для ускорения набора после ввода нескольких символов можно использовать комбинацию клавиш Ctrl+Space, чтобы среда разработки предложила подходящий вариант для

продолжения. Выбрав нужный вариант, нажмите Enter, чтобы вставить готовое выражение.

3.1 Добавление ресурсов

Далее нужно заменить текст на кнопках на слова **Красный**, **Жёлтый** и **Зелёный**. В предыдущих приложениях мы просто присваивали свойству **Text** нужную строку. Но на самом деле это неправильный подход, и даже среда разработки выводит предупреждающие значки у кнопок. По правилам, строки нужно хранить в строковых ресурсах. Подобный подход даёт разработчику множество преимуществ, в частности, быструю локализацию приложения. Можно считать это стандартом, которого нужно придерживаться.

Процесс создания строковых ресурсов очень прост. Переключитесь обратно в режим **Design** и выберите кнопку *buttonRed*. В окне свойств найдите свойство **text**. Рядом находится кнопка с многоточием. Щёлкните на кнопке. У вас откроется диалоговое окно **Resources**.

Нажмите на выпадающий список **Add new resource** для создания нового строкового ресурса и выберите **New String Value** (рисунок Л.1.2).

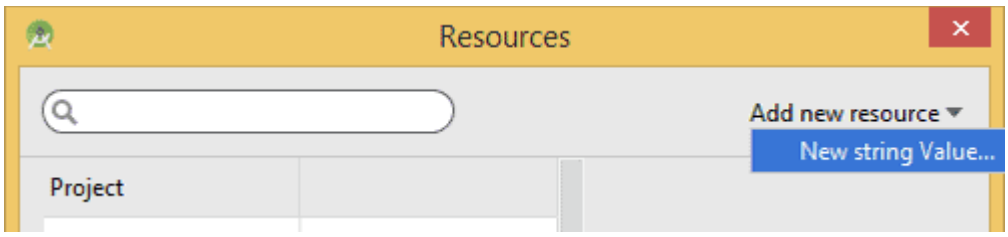

Рисунок Л.1.2 – Добавление строкового ресурса

В новом окне **New String Value Resource** в первом поле **Resource Name** введите название ресурса, например, **red**, а во втором поле **Resource Value** введите текст для кнопки (например, **Красный**). Остальные поля оставляем по умолча-

нию. Аналогичным образом поступите с другими двумя кнопками: **Жёлтый** и **Зёленый** (рисунок Л.1.3).

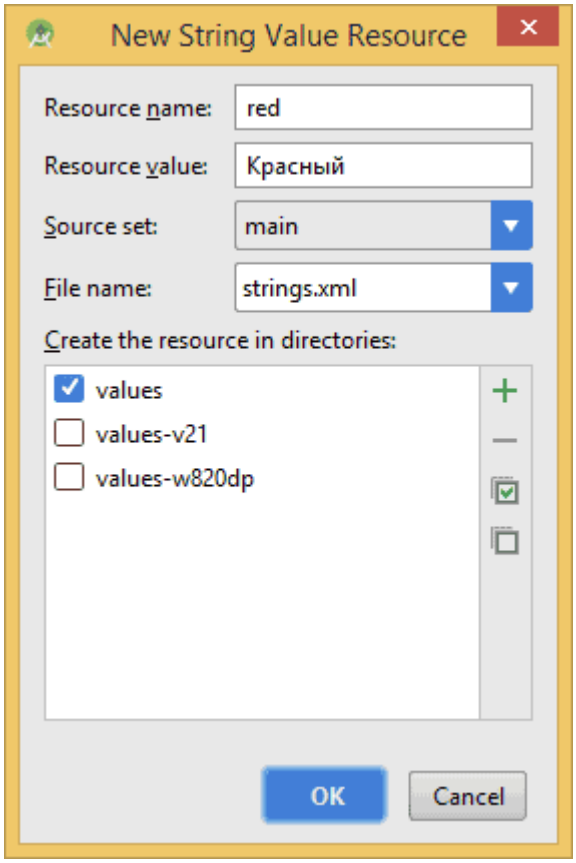

Рисунок Л.1.3 – Настройка строкового ресурса

Программно можно добиться такого же результата, отредактировав файл **strings.xml** (или **colors.xml**), который находится в папке **res/values** проекта. Сейчас он может выглядеть так.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">TrafficLight</string>
    <string name="hello_world">Hello  world!</string>
    <string name="action_settings">Settings</string>
    <string name="red">Красный</string>
    <string name="yellow">Жёлтый</string>
    <string name="green">Зелёный</string>
```

</resources>

Далее настроим элемент **TextView**. Пусть на нём выводится текст, извещающий о текущем цвете фона приложения. Так как в ресурсах у нас уже есть слова **Красный**, **Жёлтый** и **Зелёный**, изначально предназначенные для кнопок, то мы не будем создавать новые строковые ресурсы, а воспользуемся готовыми наработками. По умолчанию у нас используется красный цвет. В окне свойств выбираем свойство **text** для **TextView** и нажимаем кнопку с многоточием для вызова диалогового окна. На этот раз мы не будем щёлкать на кнопке **New Resource**, а сразу выберем строку **red**.

Переключитесь в текстовый режим и посмотрите, как теперь выглядит описание для **TextView**.

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_centerHorizontal="true"
   android:layout_centerVertical="true"
   android:text="@string/red"  />
```

Рекомендуется постоянно переключаться в этот режим и смотреть, какие изменения происходят в коде.

Далее необходимо в ресурсах задать цвета для фона программы. Ресурсы для цветов принято хранить в отдельном файле **colors.xml**, хотя технически никто не запрещает хранить их в том же файле **strings.xml**.

Откроем указанный файл и добавим ресурс жёлтого цвета между тегами **resources**:

<color name="yellowColor">#FFFF00</color>

Слева появится жёлтый квадрат, по нему легко видеть цвет заданного ресурса.

По такому же принципу добавьте зелёный цвет:

<color name="greenColor">#00FF00</color>

Самостоятельно добавьте ресурс для красного цвета под именем **redColor**.

Если нужно выбрать более сложный цвет, то проще воспользоваться мастером, как это было показано в предыдущих работах, когда выбирался розовый цвет, а затем полученный цвет скопировать в ресурсы.

Определив в ресурсах все необходимые цвета, можно сразу присвоить красный цвет для контейнера **RelativeLayout** (либо другого используемого компоновщика). В окне свойств находим для данного элемента свойство **background** (чтобы увидеть все свойства компонента, нажмите ссылку **View all properties**). Снова нажимаем кнопку с тремя точками, чтобы открыть диалоговое окно. В окне выбираем раздел **Color** и ищем ресурс **redColor** (рисунок Л.1.4).

Если посмотреть в текстовом режиме, то можно увидеть строчку **android:background="@color/redColor"** для тега **RelativeLayout**.

Общий каркас приложения завершен. У нас имеются три кнопки с соответствующими текстами, текстовая надпись со словом **Красный**, и красный фон, используемый в контейнере **RelativeLayout**. Далее будет разобрана программная логика программы. А на данном этапе можно запустить приложение, чтобы убедиться в отсутствии ошибок в разметке.

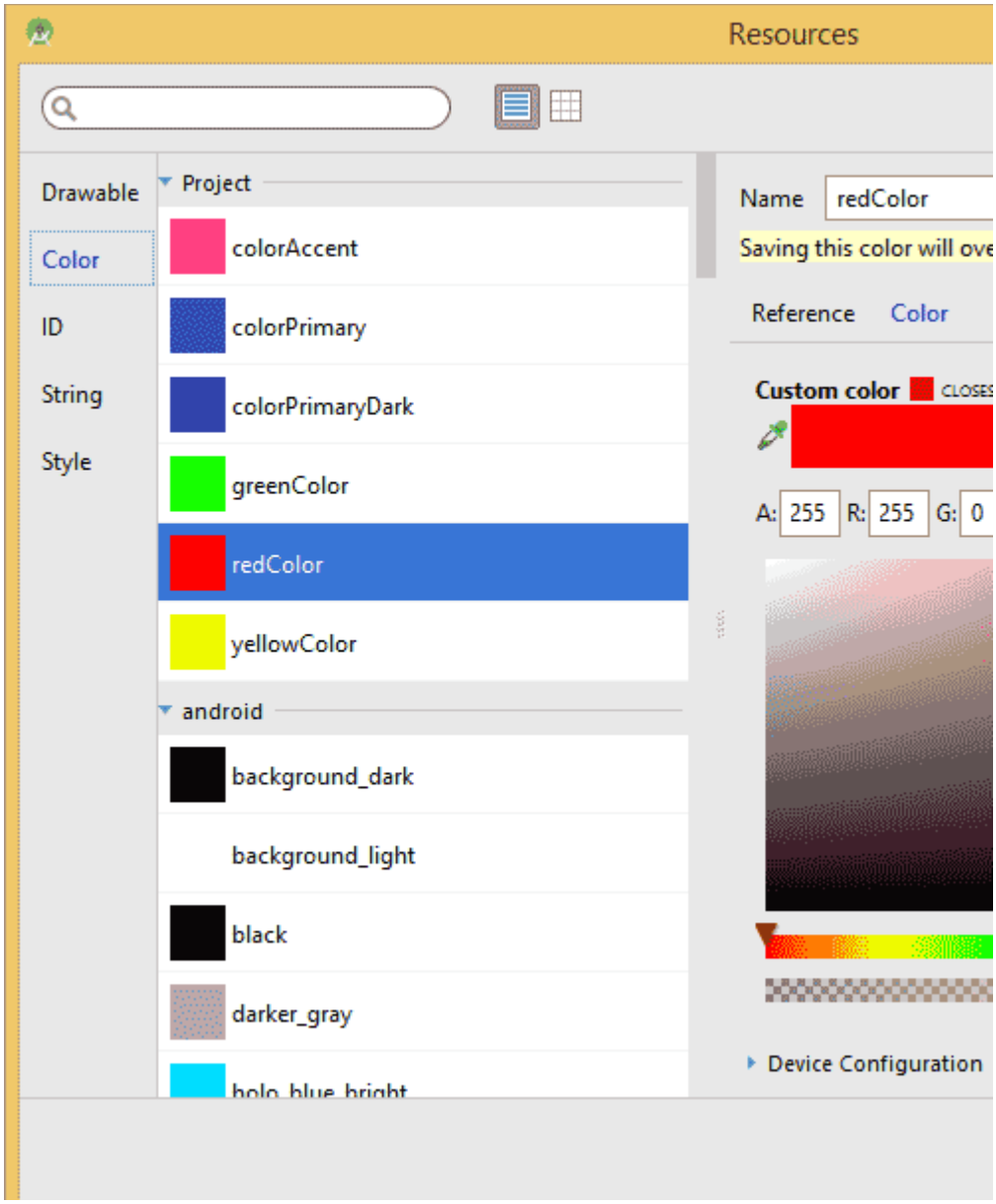

Рисунок Л.1.4 – Установка цвета фона приложения

3.2 Программный код

Основная задача при проектировании кода – обработать щелчки трёх кнопок и менять цвет фона приложения, а также текст в **TextView**. Пропишем вручную событие **onClick** в теге **Button**:

android:onClick="onRedButtonClick"

Далее в режиме **Text** помещаем курсор на названии метода и нажимаем комбинацию **Alt+Enter**, чтобы создать заготовку щелчка первой кнопки в классе **MainActivity**.

Объявим переменные в классе и получим к ним доступ в методе **onCreate()**:

```
// до метода onCreate()
private RelativeLayout mRelativeLayout;
private TextView mInfoTextView;

// в методе onCreate()
mRelativeLayout = (RelativeLayout)findViewById
                    (R.id.relativeLayout);
mInfoTextView = (TextView)findViewById(R.id.textView);
```

Если вы копируете этот код и вставляете у себя, то названия классов будут выделены красным цветом. Нажимаем комбинацию **Alt+Enter** и импортируем необходимые классы.

Также следует обратить внимание, что мы обращаемся к компонентам **RelativeLayout** и **TextView** по идентификаторам, которые ещё не создавали. Если вы ранее не задали идентификаторы для **RelativeLayout** и **TextView**, то вернитесь в файл **activity_main.xml** и добавьте атрибуты **android:id= "@+id/relativeLayout"** и **android:id="@+id/textView"**.

Пишем код для щелчка кнопки с надписью "Красный":

```
public void onRedButtonClick(View view) {
   mInfoTextView.setText(R.string.red);
   mRelativeLay-
out.setBackgroundColor(getResources().getColor(R.color.redColor)
);
}
```

Мы обращаемся к созданным ресурсам через специальный класс **R** и через точку указываем тип ресурсов и имя ресурса.

Для кнопки "Зелёный" напишите код самостоятельно, добавив метод **onGreenButtonClick()**.

Для кнопки "Жёлтый" напишем код в традиционной манере через слушателя **OnClickListener**. В методе **onCreate()** добавляем:

```
Button yellowButton = (Button)findViewById
                        (R.id.buttonYellow);
yellowButton.setOnClickListener(new  View.OnClickListener()
{
   @Override
   public void onClick(View  view) {
      mInfoTextView.setText(R.string.yellow);
      mRelativeLayout.setBackgroundColor(getResources().
                        getColor(R.color.yellowColor));
   }
});
```

В Android 6 (API 23) метод **getColor(int id)** объявили устаревшим и студия теперь подчёркивает данный метод. Можно заменить на один из двух вариантов:

```
// у второго параметра  используем  значение null
mRelativeLayout.setBackgroundColor(getResources().getColor
                (R.color.yellowColor, null));

// используем  метод из библиотеки  совместимости
mRelativeLayout.setBackgroundColor(ContextCompat.
        getColor(MainActivity.this,  R.color.yellowColor));
```

Первый вариант подойдёт, если приложение не должно поддерживать старые устройства, а сразу пишется для телефонов с API 23 и выше. Второй способ более универсальный.

Запускаем приложение и щёлкаем по кнопкам – текст в надписи и фон в приложении должны меняться в соответствии с нажатой кнопкой (рисунок Л.1.5).

Рисунок Л.1.5 – Внешний вид запущенного приложения

Полный текст кода будет выглядеть следующим образом:

```
package ru.username.trafficlight;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.RelativeLayout;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
```

```
private RelativeLayout mRelativeLayout;
private TextView mInfoTextView;

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

   mRelativeLayout = (RelativeLayout) findViewById
            (R.id.relativeLayout);
   mInfoTextView = (TextView) findViewById
            (R.id.textView);

   Button yellowButton = (Button) findViewById
            (R.id.buttonYellow);
   yellowButton.setOnClickListener(new
            View.OnClickListener() {
      @Override
      public void onClick(View view) {
         mInfoTextView.setText(R.string.yellow);
         mRelativeLayout.setBackgroundColor (ContextCom-
            pat.getColor(MainActivity.this,
            R.color.yellowColor));
      }
   });
}

public void onRedButtonClick(View view) {
   mInfoTextView.setText(R.string.red);
   mRelativeLayout.setBackgroundColor (ContextCom-
            pat.getColor(MainActivity.this,
            R.color.redColor));
}

public void onGreenButtonClick(View view) {
```

```
        mInfoTextView.setText(R.string.green);
        mRelativeLayout.setBackgroundColor (ContextCom-
            pat.getColor(MainActivity.this,
            R.color.greenColor));
    }
}
```

При желании можно упростить код для трёх кнопок, создав для них общий метод **onClick()**.

3.3 Настройка значков приложения

Начиная с Android 5.0 стиль оформления приложений изменился, и значка в заголовке приложения уже нет (хотя его можно туда поместить). Но значки всё равно нужны для отображения программы на домашнем экране или в лаунчерах.

По умолчанию студия использует изображение зелёного робота в качестве значка программы. Откройте в студии папку **res/mipmap** (раньше значки хранились в папке **res/drawable**, но принцип сохранился). Эта папка является виртуальной, в действительности существуют папки **res/drawable-hdpi**, **res/drawable-mdpi**, **res/drawable-xhdpi**, **res/drawable-xxhdpi**. В каждой из этих папок есть файл с одинаковым именем **ic_launcher.png**. Вся разница между этими файлами заключается в размерах. В зависимости от разрешения экрана на устройстве система выбирает наиболее подходящее по размеру изображение и выводит его в качестве значка в заголовке приложения и на домашнем экране. Самый простой вариант заменить стандартное изображение на своё – создать своё изображение и заменить им имеющийся значок. Рекомендуется создавать под каждое разрешение свой значок. Причем здесь указаны не все варианты. В таком случае вам нужно создать самостоятельно папку, например, **drawable-xxxhdpi** и разместить там картинку требуемого размера. Если вы пропустите какой-то размер, то система попробует взять какой-нибудь значок с этим

именем из другой папки и смасштабировать его. Но так делать не рекомендуется.

Если вы не хотите менять существующие стандартные значки, а хотите использовать значки под другим именем, то в этом случае нужно подготовить все необходимые размеры, разместить их во всех папках **drawable-** под своим именем, а затем в манифесте заменить строчку у атрибута **android:icon**:

```
<application
   android:allowBackup="true"
   android:icon="@drawable/ic_launcher_cat"
   android:label="@string/app_name"
   android:theme="@style/AppTheme" >
```

В новых версиях это будет строчка **android:icon="@mipmap/ic_launcher"**, принципиальной разницы нет.

В состав студии входит набор предопределённых значков и генератор собственных значков. Чтобы его увидеть на экране, можно щёлкнуть правой кнопкой на папке **drawable** или **mipmap** и выбрать в меню **New | Image Asset**.

Откроется диалоговое окно, где вы можете указать в качестве источника файл на компьютере, вариант из клипарта или набор символов. Также можно задать форму значка, цвет фона и прочие параметры.

Кроме значков для различных разрешений, генератор создаст дополнительный файл с суффиксом **-web**, который будет скопирован в папку **main**. Этот файл используется для Google Play при размещении в нем приложения и составлении описания.

4 Контрольные вопросы

1. Опишите процесс программного создания элемента управления.

2. Как создаются строковые ресурсы? Какие параметры можно при этом настроить?

3. В каком файле хранятся ресурсы для цветов?

4. Перечислите известные вам способы создания ресурсов для цветов.

5. Как изменить идентификатор элемента управления?

6. Как изменить значок приложения?

5 Задание для самостоятельного выполнения

Разработайте простейший вариант игры «Пятнашки».

Начальная конфигурация игрового поля может задаваться пользователем (а может – случайным образом). Обеспечить подсветку ячеек поля, доступных для следующего хода, а также ячейки с последним сделанным ходом.

Пользовательский интерфейс можно реализовать на основе компоновщика TableLayout. Ячейки игрового поля можно задать с помощью иконок либо с помощью кнопок.

Лабораторная работа №2. Переключение между экранами приложения

1 Цель работы

Целью работы – ознакомиться с общими принципами разработки приложений с несколькими активностями, получит практические навыки создания дополнительных активностей, переключения между активностями, а также передачи данных из одной активности в другую.

2 Содержание отчета

- титульный лист;
- краткое теоретическое описание;
- задание на лабораторную работу, включающее математическую формулировку задачи;
- результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений;

3 Порядок выполнения работы

3.1 Простое переключение на другой экран

Приложение не всегда состоит из одного экрана. Например, при нажатии на кнопку «О программе» пользователь скорее всего попадет на новый экран, где находится информация о версии программы, авторе, адресе сайта и т.д. Можно воспринимать экран активности как веб-страницу со ссылкой на другую страницу. Если посмотреть на код в файле **MainActivity.java** из прошлых работ, то можно увидеть, что класс **MainActivity** тоже относится к **Activity** (или его наследникам) или, если говорить точнее, наследуется от него.

public class MainActivity extends AppCompatActivity

Как нетрудно догадаться, следует создать новый класс, который может быть похож на **MainActivity** и затем каким-то образом переключиться на него при нажатии кнопки.

Для примера откройте приложение из второй практической работы и будем использовать для опытов кнопку (или создайте новый проект с одной кнопкой на экране). Далее создадим новую форму для отображения некой информации.

Для лучшего понимания создавать новую активность будем вручную, хотя в студии есть готовые шаблоны.

Создадим новый XML-файл разметки **activity_about.xml** в папке **res/layout**. Щёлкните правой кнопкой мыши на папке **layout** и выберите из контекстного меню **New | Layout resource file**. Появится диалоговое окно. В первом поле вводим имя файла **activity_about**. Во втором нужно ввести корневой элемент. По умолчанию там стоит **LinearLayout**. Стираем текст и вводим **ScrollView** (рисунок Л.2.1).

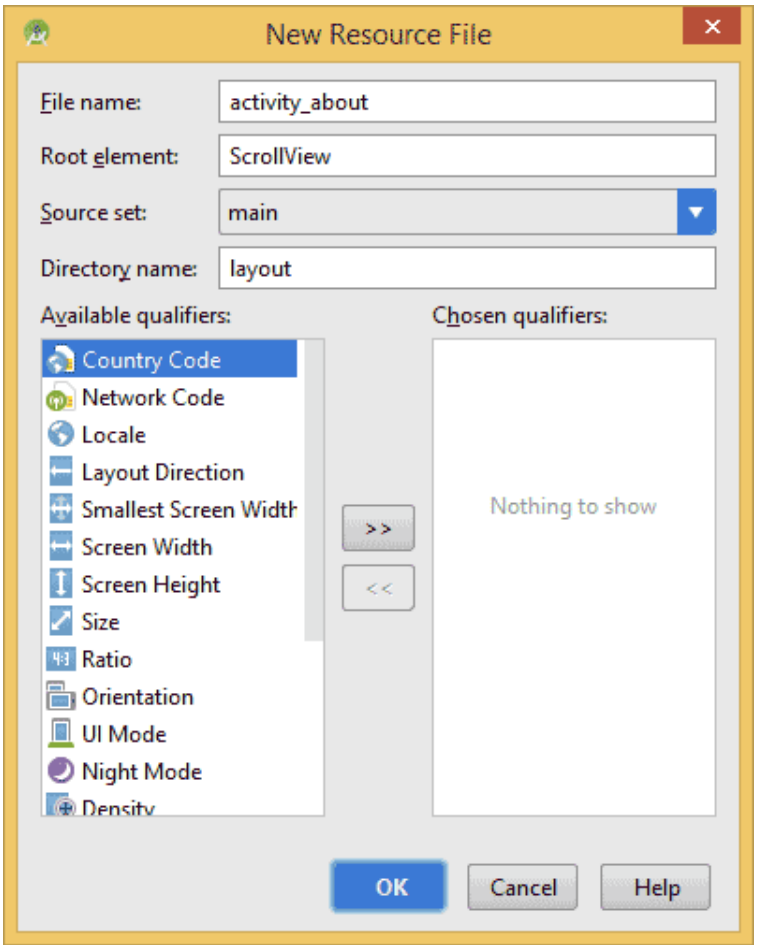

Рисунок Л.2.1 – Добавление новой активности

Получится соответствующая заготовка, в которую вставим элемент **TextView**.

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView_about_content"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/about_text"/>

</ScrollView>
```

Информация будет извлекаться из ресурсов, а именно из строкового ресурса **about_text**. Сейчас он подсвечен красным цветом, сигнализируя об отсутствии информации. Можно было нажать **Alt+Enter** и ввести текст в диалоговом окне. Но для нашего примера этот способ не подойдёт, так как наш текст будет многострочным, с использованием управляющих символов. Поэтому поступим другим способом. Откроем файл **res/values/strings.xml** и введем следующий текст вручную:

```
<string name="about_text">
    У лукоморья дуб зелёный;\n
    Златая цепь на дубе том:\n
    И днём и ночью <b>кот учёный</b>\n
    Всё ходит по цепи кругом;\n
    Идёт <b>направо</b> - песнь заводит,\n
    <b>Налево</b> - сказку говорит.</string>
```

Здесь использованы простейшие HTML-теги форматирования текста типа <b>, <i>, <u>. В нашем случае выделены жирным слова, которые относятся к коту и направлению движения. Для перевода текста на новую строку служат символы **\n**. Добавим ещё один строковый ресурс для заголовка нового экрана:

<string name="about_title">О программе</string>

Далее необходимо создать класс для окна **AboutActivity.java**. Выбираем в меню **File | New | Java Class** (если данного пункта меню нет, переключитесь на файл **MainActivity.java** и попробуйте снова) и заполняем нужные поля. На первых порах достаточно указать только имя (рисунок Л.2.2).

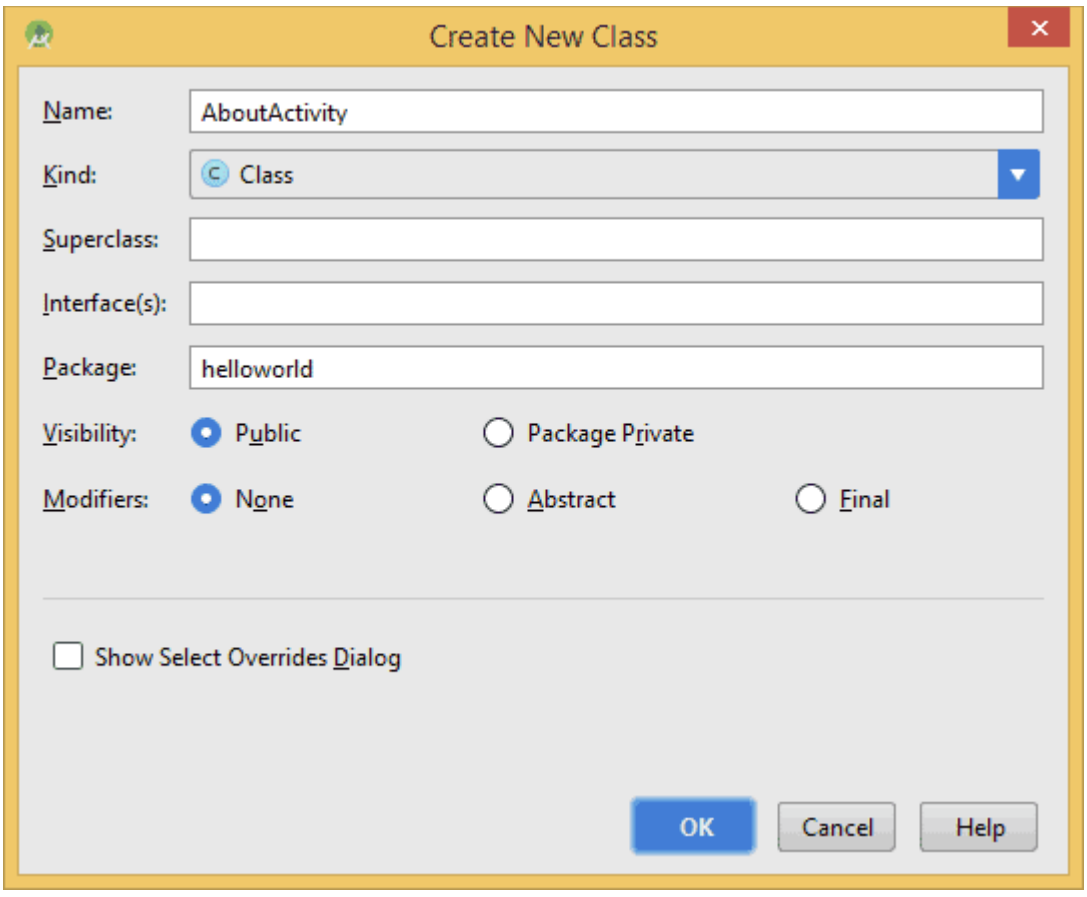

Рисунок Л.2.2 – Создание класса активности

Получаем заготовку класса. Сейчас класс практически пустой. Добавим код вручную. Класс должен наследоваться от абстрактного класса **Activity** или его родственников типа **FragmentActivity**, **AppCompatActivity** и т.д. Дописываем **extends AppCompatActivity**. У класса активности должен быть метод **onCreate()**. Ставим курсор мыши внутри класса и выбираем в меню **Code | Override Methods** (Ctrl+O). В диалоговом окне ищем метод **onCreate()**, можно набирать на клавиатуре первые символы для быстрого поиска. В созданном методе нужно вызвать метод **setContentView()**, который подгрузит на экран подготовленную разметку. В итоге получится такой вариант:

```
package ru.username.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class AboutActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_about);
    }
}
```

Далее наша задача – перейти на новый экран при щелчке кнопки на первом экране. Возвращаемся обратно к классу **MainActivity**. Напишем обработчик щелчка кнопки:

```
public void onClick(View view) {
    Intent intent = new Intent(MainActivity.this,
                AboutActivity.class);
```

```
    startActivity(intent);
}
```

Для запуска нового экрана необходимо создать экземпляр класса **Intent** и указать в первом параметре текущий класс, а во втором – класс для перехода, у нас это **AboutActivity**. После этого вызывается метод **startActivity()**, который и запускает новый экран.

Если сейчас попытаться проверить работу приложения в эмуляторе, то мы получим сообщение об ошибке, т. к. мы пропустили один важный шаг. Необходимо зарегистрировать новый **Activity** в манифесте **AndroidManifest.xml**. Найдите этот файл в своем проекте и дважды щёлкните на нём. Откроется окно редактирования файла. Добавьте новый тег **<activity>** после закрывающего тега **</activity>** для первой активности. Получится следующее:

```
<activity android:name=".AboutActivity"
    android:label="@string/about_title">
</activity>
```

Здесь нам пригодился строковый ресурс **about_title**. Запускаем приложение, щёлкаем на кнопке и получаем окно **О программе** (рисунок Л.2.3).

После вызова метода **startActivity()** запустится новая активность (в данном случае **AboutActivity**), она станет видимой и переместится на вершину стека, содержащего работающие компоненты. При вызове метода **finish()** из новой активности (или при нажатии аппаратной клавиши возврата) она будет закрыта и удалена из стека. Разработчик также может перемещаться к предыдущей (или к любой другой) активности, используя всё тот же метод **startActivity()**.

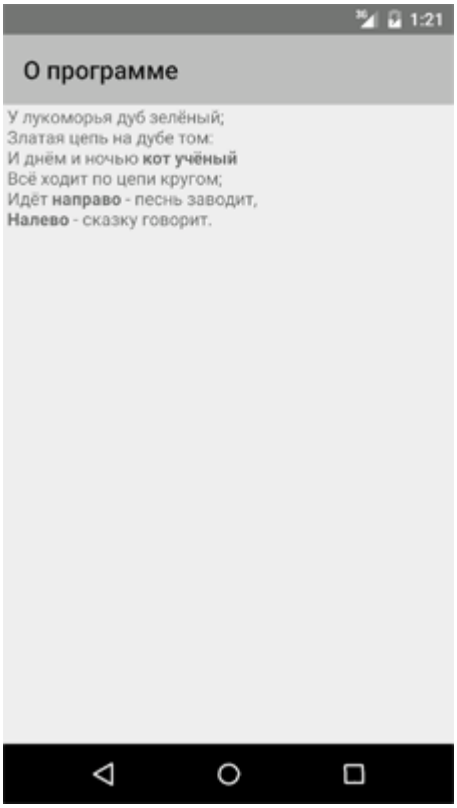

Рисунок Л.2.3 – Окно «О программе»

Существует и более простой способ создания новых активностей. Для этого выберите в меню **File | New | Activity | Empty Activity** (или другой шаблон). Появится уже знакомое окно создания новой активности. Заполняем необходимые поля.

Нажимаем на кнопку **Finish,** и активность будет готова. Чтобы убедиться в этом, откройте файл манифеста и проверьте наличие новой записи. Файлы класса и разметки также сгенерируются автоматически.

На первых порах можно посоветовать вручную создавать все необходимые компоненты для новой активности, чтобы понимать взаимосвязь между классом, разметкой и манифестом. А уже в дальнейшем можно использовать мастер создания активности для ускорения работы.

3.2 Передача данных между активностями

Мы использовали простейший пример для вызова другого экрана активности. Иногда требуется не только вызвать новый экран, но и передать в него данные. Например, имя пользователя. В этом случае нужно задействовать специальную область **extraData**, который имеется у класса **Intent**.

Область **extraData** - это список пар *ключ/значение*, который передаётся вместе с намерением. В качестве ключей используются строки, а для значений можно использовать любые примитивные типы данных, массивы примитивов, объекты класса **Bundle** и др.

Для передачи данных в другую активность используется метод **putExtra()**:

```
intent.putExtra("Ключ", "Значение");
```

Принимающая активность должна вызвать какой-нибудь подходящий метод: **getIntExtra()**, **getStringExtra()** и т.д.:

```
int count = getIntent().getIntExtra("name", 0);
```

Создайте новый проект с двумя активностями. У первой активности разместим два текстовых поля и кнопку. Внешний вид приложения может быть следующим (рисунок Л.2.4):

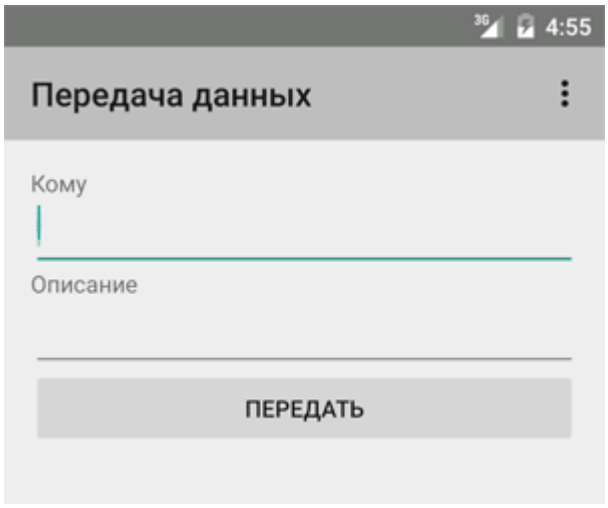

Рисунок Л.2.4 – Внешний вид приложения для передачи данных

У второй активности **SecondActivity** установим элемент **TextView**, в котором будем выводить текст, полученный от первой активности. Напишем следующий код для метода **onCreate()** у второй активности.

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

   String user = "Пользователь";
   String gift = "НИЧЕГО";

   TextView infoTextView =
(TextView)findViewById(R.id.textViewInfo);
   infoTextView.setText(user + " , вам передали " + gift);
}
```

Если сейчас запустить программу и просто вызвать второе окно, как это было описано в первой части статьи, то мы увидим надпись по умолчанию **Пользователь, вам передали НИЧЕГО**.

Добавляем код у первой активности:

```
public void onClick(View view) {
   EditText userEditText = (EditText) findViewById
            (R.id.editTextUser);
   EditText giftEditText = (EditText) findViewById
            (R.id.editTextGift);

   Intent intent = new Intent(MainActivity.this,
            SecondActivity.class);

//в ключ username заносим текст из первого текстового поля
```

```
        intent.putExtra("username",  userEditText.getText().
                toString());
        // в ключ gift заносим текст из второго текстового поля
        intent.putExtra("gift",  giftEditText.getText().toString());
        startActivity(intent);
    }
```

Мы поместили в специальный контейнер объекта **Intent** два ключа со значениями, которые берутся из текстовых полей. Когда пользователь введёт данные в текстовые поля, они попадут в этот контейнер и будут переданы второй активности (рисунок Л.2.5).

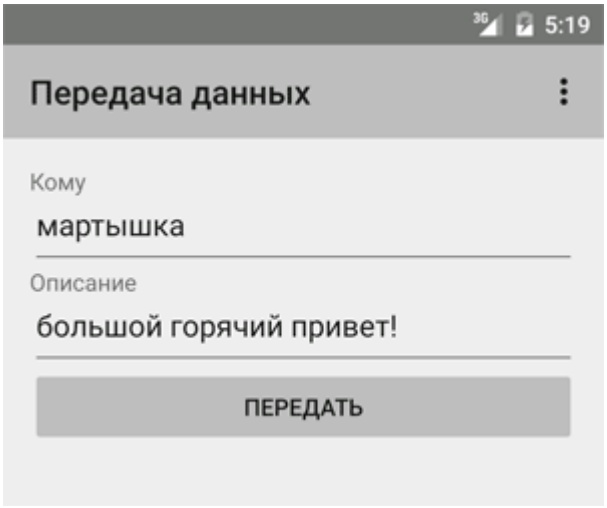

Рисунок Л.2.5 – Ввод данных для передачи в другую активность

Вторая активность должна быть готова к приёму сообщений следующим образом (выделено жирным).

```
// Значения по умолчанию
String user = "Пользователь";
String gift = "НИЧЕГО";

user = getIntent().getExtras().getString("username");
gift = getIntent().getExtras().getString("gift");
```

```
TextView infoTextView = (TextView)findViewById
        (R.id.textViewInfo);
infoTextView.setText(user + " , вам передали " + gift);
```

Результат может выглядеть примерно следующим образом (рисунок Л.2.6).

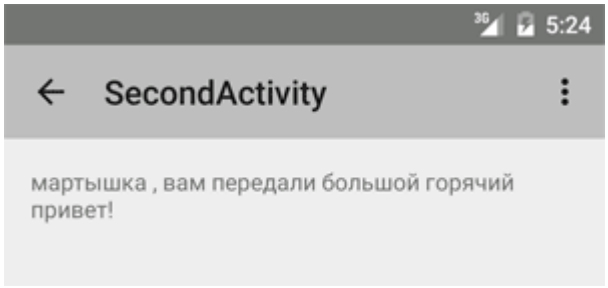

Рисунок Л.2.6 – Результат работы приложения

В сложных примерах желательно добавить проверку при обработке данных. Возможны ситуации, когда вторая активность запускается с пустыми данными типа **null**, что может привести к краху приложения.

В нашем случае мы знаем, что ждём строковое значение, поэтому код можно переписать так:

```
Intent intent = getIntent();
user = intent.getStringExtra("username");
```

Или так:

```
user = getIntent().getStringExtra("username");
```

У программы есть недостаток – не понятно, от кого мы получаем приветы. Поэтому в качестве дальнейшего развития риложения можно добавить ещё одно текстовое поле для ввода имени пользователя, который отправляет сообщение.

3.3 Получение результата из другой активности

Не всегда бывает достаточно просто передать данные другой активности. Иногда требуется получить информацию обратно от другой активности при её закрытии. Если раньше мы

использовали метод **startActivity(Intent intent)**, то существует родственный ему метод **startActivityForResult(Intent intent, int RequestCode)**. Разница между методами заключается в дополнительном параметре **RequestCode**. По сути это просто целое число, которое нужно для того, чтобы различать, от кого пришёл результат. Допустим, у нас есть пять дополнительных экранов, мы присваиваем им значения от 1 до 5, и по этому коду можно будет определить, чей результат нужно обрабатывать. Можно также использовать значение -1, тогда это будет равносильно вызову метода **startActivity()**, т.е. никакого результата не получим.

Если вы используете метод **startActivityForResult()**, то вам необходимо переопределить в коде метод для приёма результата **onActivityResult()** и обработать полученный результат.

Разберём шуточный пример. Предположим, вы сыщик. Поступила информация, что в ресторане со стола влиятельного человека украли два кусочка колбасы и другие продукты. Подозрение пало на трёх подозреваемых - ворона, пёсик и кот Васька. Необходимо выявить имя воришки.

Создаём новый проект **Sherlock** с двумя активностями. На первом экране будет кнопка для переключения на второй экран и текстовая метка, в которой будет отображено имя воришки.

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="5dp" >

    <ImageView
        android:id="@+id/icon"
        android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:adjustViewBounds="true"
            android:background="#fa2255"
            android:scaleType="centerCrop"
            android:src="@drawable/sherlock"  />

        <TextView
            android:id="@+id/textViewLabel"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/who"
            an-
droid:textAppearance="?android:attr/textAppearanceMedium"  />

        <TextView
            android:id="@+id/textViewInfo"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            an-
droid:textAppearance="?android:attr/textAppearanceLarge"  />

        <Button
            android:id="@+id/buttonChoose"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:onClick="onClick"
            android:text="Сделать  выбор" />

    </LinearLayout>
```

На втором экране будет группа переключателей:

```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
        android:id="@+id/LinearLayout1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:padding="10dip" >

        <TextView
            android:id="@+id/textView1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Выберите правильный ответ"
            an-
droid:textAppearance="?android:attr/textAppearanceMedium" />

        <RadioGroup
            android:id="@+id/radioGroup1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" >

            <RadioButton
                android:id="@+id/radioCrow"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:checked="false"
                android:onClick="onRadioClick"
                android:text="Ворона" />

            <RadioButton
                android:id="@+id/radioDog"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:onClick="onRadioClick"
                android:text="Пёсик" />

            <RadioButton
```

```
            android:id="@+id/radioCat"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onRadioClick"
            android:text="Кот Васька" />

    </RadioGroup>

</LinearLayout>
```

Так как мы будем ожидать ответ из второго экрана, то нам необходимо задействовать метод **startActivityForResult()** на первом экране, в котором мы передадим переменную **CHOOSE_THIEF** в качестве параметра **RequestCode**.

```
static final private int CHOOSE_THIEF = 0;

public void onClick(View v) {
    Intent questionIntent = new Intent(MainActivity.this,
                    ChooseActivity.class);
    startActivityForResult(questionIntent, CHOOSE_THIEF);
}
```

Обратите внимание на код. При щелчке на кнопке мы собираемся работать со вторым экраном **ChooseActivity** и запускаем второй экран с ожиданием результата.

Перейдем на второй экран и напишем код для второй активности.

```
public final static String THIEF =
                "ru.username.sherlock.THIEF";

public void onRadioClick(View v) {
    Intent answerIntent = new Intent();
```

```
    switch (v.getId()) {
    case R.id.radioDog:
            answerIntent.putExtra(THIEF, "Пёсик");
            break;
    case R.id.radioCrow:
            answerIntent.putExtra(THIEF, "Ворона");
            break;
    case R.id.radioCat:
            answerIntent.putExtra(THIEF, "Кот Васька");
            break;

    default:
            break;
    }

    setResult(RESULT_OK, answerIntent);
    finish();
}
```

В данном случае, когда сыщик выбирает имя преступника, через метод **putExtra()** передаётся имя ключа и его значение.

Для удобства, после выбора мы сразу закрываем второе окно и перед закрытием передаём значение **RESULT_OK**, чтобы было понятно, что выбор сделан. Если пользователь закроет экран через кнопку Back, то будет передано значение **RESULT_CANCELED**.

Метод **setResult()** принимает два параметра: результирующий код и сам результат, представленный в виде намерения. Результирующий код говорит о том, с каким результатом завершилась работа активности. Как правило, это либо **Activity.RESULT_OK**, либо **Activity.RESULT_CANCELED**. В некоторых случаях нужно использовать собственный код возврата для обработки специфических для приложения вариантов. Метод **setResult()** поддерживает любое целочисленное значение.

Если данные передаются явно через кнопку, то следует добавить метод **finish()**, чтобы закрыть вторую активность за ненадобностью. Если переход происходит через кнопку Назад, то это делать не обязательно.

Если активность была закрыта пользователем при нажатии аппаратной кнопки возврата или если метод **finish()** был вызван раньше, чем метод **setResult()**, результирующий код установится в **RESULT_CANCELED**, а возвращенное намерение покажет значение **null**.

Возвращаемся на первый экран. Первый экран ожидает ответа от второго экрана, поэтому нужно добавить в код метод **onActivityResult()**.

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
        Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    TextView infoTextView = (TextView) findViewById
        (R.id.textViewInfo);

    if (requestCode == CHOOSE_THIEF) {
        if (resultCode == RESULT_OK) {
            String thiefname = data.getStringExtra
                (ChooseActivity.THIEF);
            infoTextView.setText(thiefname);
        }
        else {
            infoTextView.setText(""); // стираем текст
        }
    }
}
```

Метод ожидает входящие данные с кодом **CHOOSE_THIEF**, и если такие данные поступят, то извлекает

значение из ключа **ChooseActivity.THIEF** с помощью метода **getStringExtra**. Полученное значение выводится в **TextView** (переменная **infoTextView**). Если мы вернулись на экран через кнопку Back, то просто стираем текст.

При закрытии дочерней активности внутри родительского компонента срабатывает обработчик **onActivityResult()**. Обработчик **onActivityResult()** принимает несколько параметров.

- код запроса. Это код, который использовался для запуска активности, возвращающей результат;
- результирующий код. Это код результата, устанавливаемый дочерней активностью и указывающий, как завершилась её работа. Это может быть любое целочисленное значение, но, как правило, это либо **Activity.RESULT_OK**, либо **Activity. RESULT_CANCELED;**
- данные. Это намерение, используемое для упаковки возвращаемых данных. В зависимости от назначения дочерней активности оно может включать путь URI, представляющий выбранную часть содержимого. В качестве альтернативы (или дополнения) дочерняя активность может возвращать информацию в виде простых значений, упакованных в параметр намерения **extras.**

Если работа дочерней активности завершилась непредвиденно или если перед её закрытием не был указан код результата, этот параметр станет равен **Activity.RESULT_CANCELED**.

Запускаем проект, нажимаем на кнопку и переходим на второй экран. Там выбираем один из вариантов. Если выбрать ворону, то экран закроется и имя преступника отобразится на первом экране (рисунок Л.2.7).

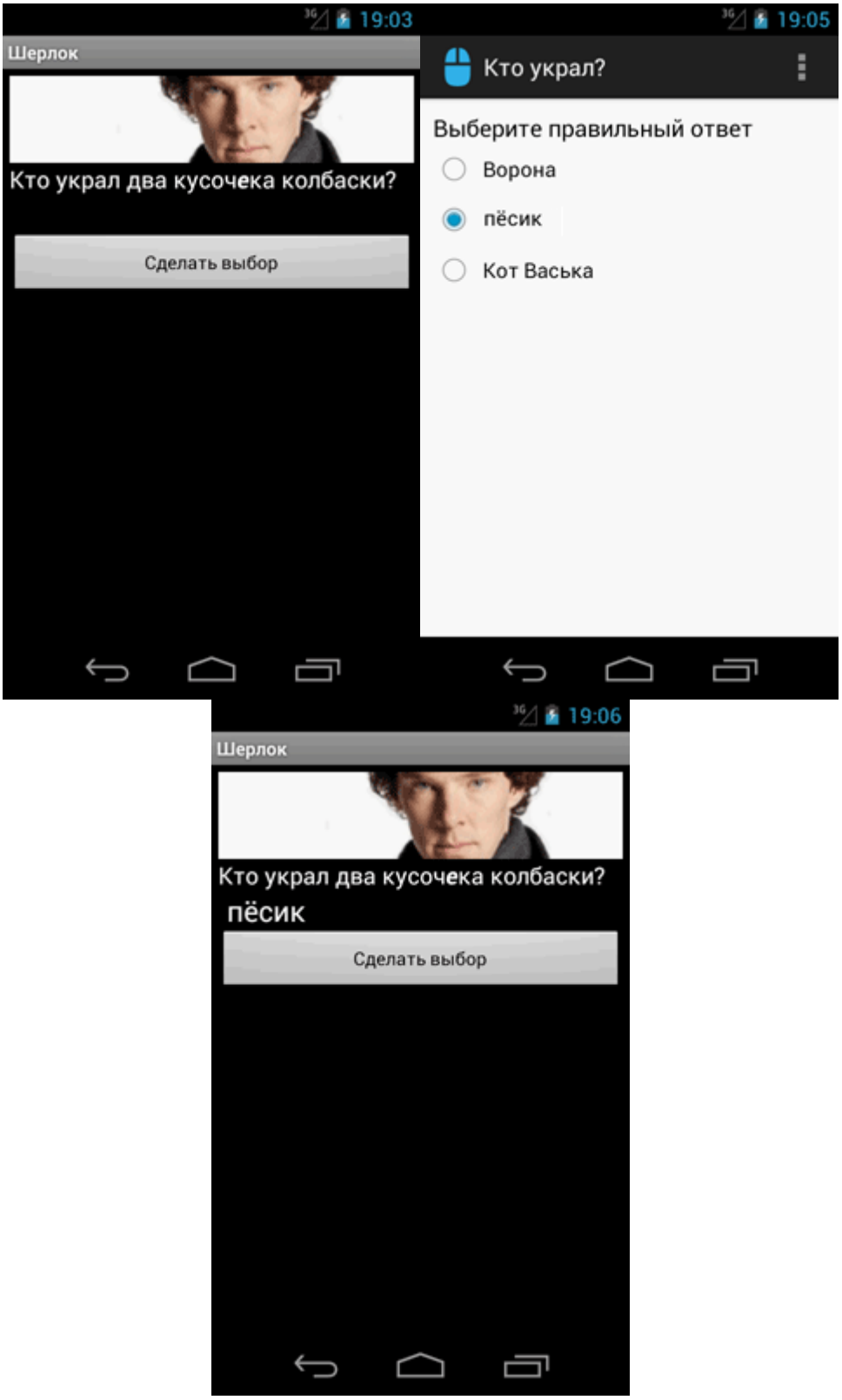

Рисунок Л.2.7 – Результаты работы программы

3.4 Использование фильтров

В предыдущих разделах был показан распространённый способ перехода на другую активность, когда в методе **startActivity()** указывается текущий класс и класс для перехода. При этом класс активности не обязательно должен быть частью приложения. Если известно имя класса из другого приложения, то можно перейти и на него. Однако имеется и другой способ перехода в другую активность.

На практике он встречается реже, но иногда может оказаться полезным. Допустим, у нас уже имеется вторая активность. В манифесте добавим к ней специальный фильтр:

```
<activity
   android:name=".SecondActivity"
   android:label="@string/title_activity_second" >
   <intent-filter >
      <action android:name="ru.username.testapplication.
            SecondActivity" />
      <category android:name="android.intent. catego-
            ry.DEFAULT" />
   </intent-filter>
</activity>
```

Далее запускаем вторую активность через щелчок кнопки следующим способом:

```
public void onClick(View view) {
   startActivity(new Intent("ru.username.testapplication.
            SecondActivity"));
}
```

Заменим длинную строку на константу:

```
public static final String ACTION_SECOND_ACTIVITY =
        "ru.username.testapplication.SecondActivity";
```

```
public void onClick(View view) {
   startActivity(new Intent(ACTION_SECOND_ACTIVITY));
}
```

Разберем вышеописанные действия подробнее. Для второй активности мы прописали фильтр и указали имя для **action** в атрибуте **android:name**. Для удобства мы просто поместили в него полное имя активности с названием пакета. Конструктор класса **Intent** имеет несколько перегруженных версий. В одной из версий можно указать строку для действия. Мы указали своё созданное действие, которое прописано у второй активности. Система во время работы просматривает манифесты всех установленных приложений. При поиске соответствия система находит наш фильтр и запускает нужную активность.

Имя категории фильтра **android.intent.category.DEFAULT** говорит системе, что следует выполнить действие по умолчанию, а именно, запустить активность. Существует и другие имена, которые пока что не рассматриваются.

Разберем далее, что произойдёт, если создать ещё одну активность и указать такой же фильтр, как у второй активности? Создайте в приложении третью активность и скопируйте в нее блок с фильтром от второй активности.

```
<activity
   android:name=".ThirdActivity"
   android:label="@string/title_activity_third" >
   <intent-filter>
      <action android:name="ru.username.testapplication.
            SecondActivity" />
      <category android:name="android.intent.category.
            DEFAULT" />
   </intent-filter>
</activity>
```

Щёлкаем по кнопке в первой активности. Система попросит выбрать нужный вариант (рисунок Л.2.8).

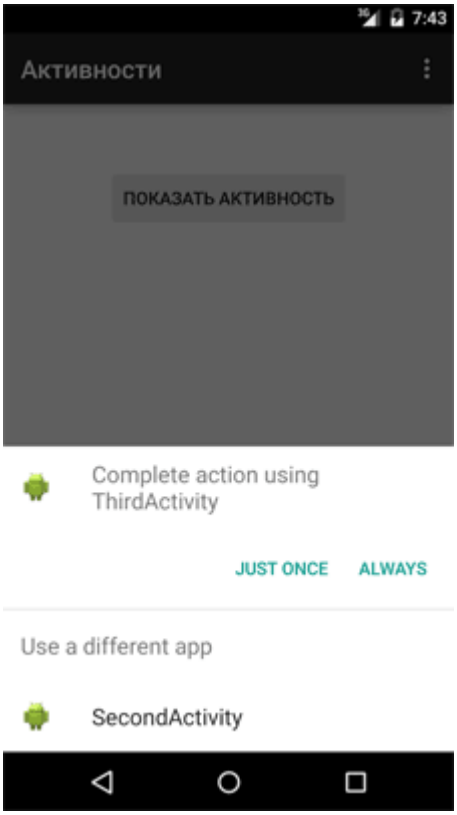

Рисунок Л.2.8 – Окновыбора активности для отображения

Если выбрать пункт **ALWAYS**, то в следующий раз выбирать не придётся. Чтобы сбросить выбор, следует зайти в свойства приложения в Настройках и нажать кнопку **Clear defaults**.

4 Контрольные вопросы

1. Как создать новую активность в мобильном приложении?
2. Для каких целей используются экземпляры класса Intent?
3. Какой метод служит для запуска дополнительных активностей?
4. Как осуществляется передача данных между активностями?

5. Как получить информацию от дополнительной активности при ее закрытии?

6. Как перейти на другую активность, используя фильтры?

5 Задание для самостоятельного выполнения

Модифицируйте любое приложение практической работы №2 путем добавления в него дополнительной активности «Настройки». Предусмотреть в ней возможность изменения внешнего вида компонентов главной активности (цвета фона или переднего плана компонентов, шрифта, заголовка активности и т. п. – не менее трех различных настроек). По нажатию кнопки «Применить» должен происходить возврат к главной активности с применением сделанных настроек. При повторном вызове активности «Настройки» пользовательский выбор должен сохраняться.

Лабораторная работа №3. Отслеживание ориентации мобильных устройств

1 Цель работы

Цель работы – получение практических навыков отслеживания ориентации устройств при разработке мобильных кроссплатформенных приложений.

2 Содержание отчета

- титульный лист;
- краткое теоретическое описание;
- задание на лабораторную работу, включающее математическую формулировку задачи;
- результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений;

3 Порядок выполнения работы

Когда создавались первые портативные устройства – КПК и смартфоны, то за основу бралась настольная операционная система и модифицировалась под мобильное устройство. Лишние функции удалялись, а некоторые функции добавлялись. Но при этом как-то разработчики зачастую упускали из виду, что в отличие от громоздких мониторов и экранов ноутбуков, карманные устройства можно вращать в руках. Первые устройства не умели менять ориентацию экрана. Некоторые программисты самостоятельно стали создавать программы, которые умели переключаться в разные режимы. Затем эту возможность стали включать в настройки аппарата. Позже аппараты научились самостоятельно определять ориентацию экрана.

Всего существует два режима – портретный и альбомный. На большинстве телефонов используется по умолчанию портретный режим (как на паспорте). Альбомный режим знаком нам по обычным мониторам.

3.1 Создание альтернативной разметки

Рассмотрим следующий случай. Предположим, у нас в приложении имеется одно текстовое поле и шесть кнопок (рисунок Л.3.1).

Рисунок Л.3.1 – Внешний вид приложения (портретный режим)

Но стоит повернуть устройство на 90 градусов (для эмулятора нужно нажать комбинацию клавиш Ctrl+F11), как сразу обнаруживаются проблемы. Пятая кнопка видна частично, а шестая вообще оказалась за пределами видимости (рисунок Л.3.2).

Чтобы избежать такой проблемы, необходимо как-то по-другому скомпоновать кнопки. Например, расположить их не подряд друг за другом, а разбить на пары. Воспользуемся контейнером **TableLayout**. С его помощью мы можем разбить кнопки на две колонки и поместить их в три ряда.

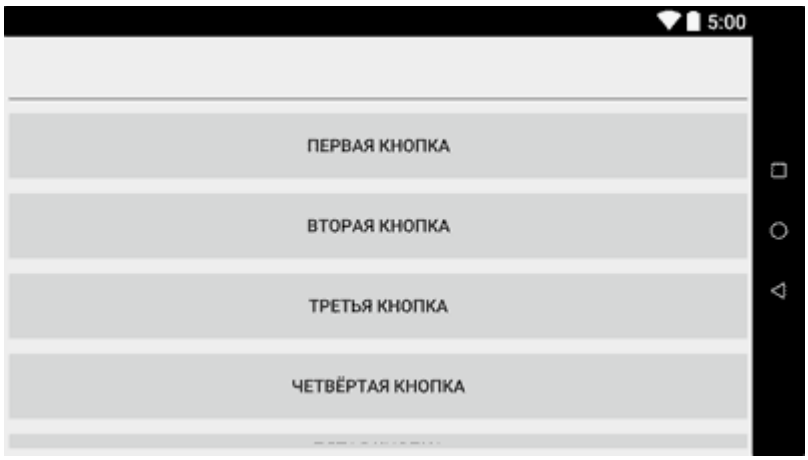

Рисунок Л.3.2 – Внешний вид приложения в альбомном режиме (неудачная компоновка)

Для этой операции понадобится сделать несколько важных шагов. Сначала нужно создать новую подпапку в папке **res**. Выделяем папку **res**, вызываем из него контекстное меню и последовательно выбираем команды **New | Android resource directory** (рисунок Л.3.3).

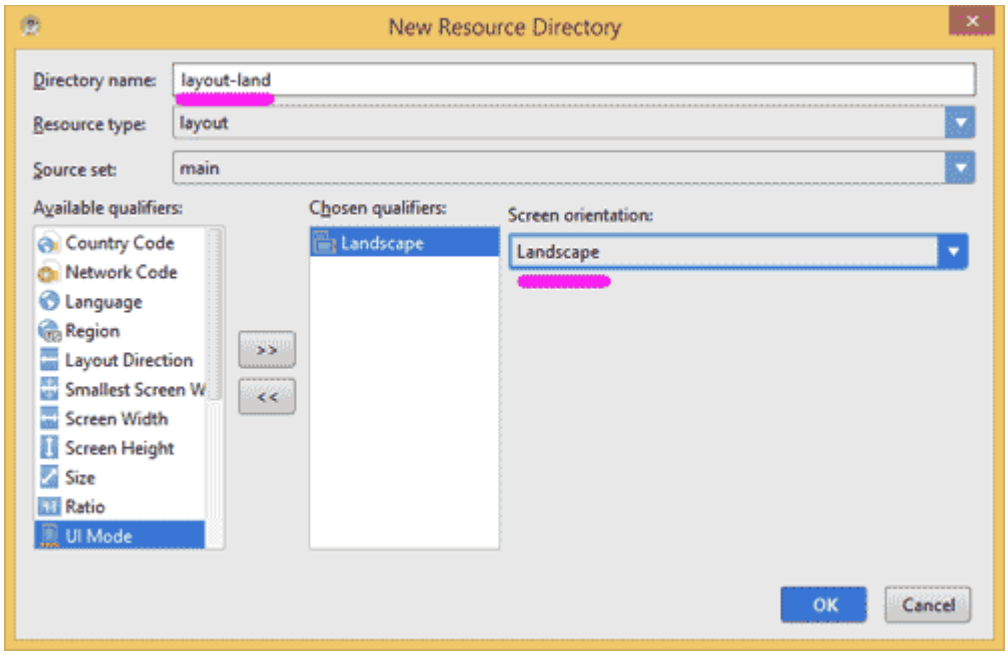

Рисунок Л.3.3 – Создание ресурса для альбомной ориентации

В диалоговом окне из выпадающего списка **Resource type:** выбираем **layout**. В списке **Available qualifiers:** находим элемент **Orientation** и переносим его в правую часть **Chosen qualifiers:** с помощью кнопки с двумя стрелками. По умолчанию появится имя папки **layout-port** в первой строке **Directory Name:**. Но нам нужен альбомный вариант, поэтому в выпадающем списке **Screen orientation** выбираем **Landscape**. Теперь название папки будет **layout-land**.

Можно обойтись без помощи мастера, создав папку сразу через меню **New | Directory**. Этот способ годится для опытных разработчиков, которые знают, как следует назвать папку. Важно запомнить, что имя даётся не произвольно, а именно в таком виде **layout-land**. По суффиксу **-land** система понимает, что речь идет о новом режиме. Теперь осталось создать в данной папке новый XML-файл **activity_main.xml**. Вызываем контекстное меню у папки **layout-land** и выбираем команды **New | Layout Resource File**. В диалоговом окне присваиваем имя **activity_main.xml**, которое должно совпадать с именем существующего файла. Во втором поле вводим **LinearLayout**, по мере ввода появится подсказка, облегчающая выбор (рисунок Л.3.4).

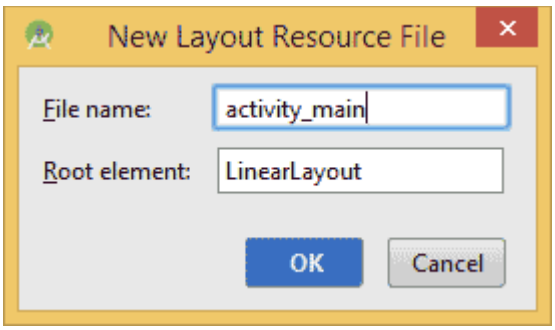

Рисунок Л.3.4 – Создание файла ресурса

Откроем созданный файл и модифицируем его следующим образом.

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

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">

   <LinearLayout
      android:orientation="vertical"
      android:layout_height="wrap_content"
      android:layout_width="match_parent"
      android:layout_gravity="center"
      android:paddingLeft="20dp"
      android:paddingRight="20dp">

      <EditText
         android:layout_height="wrap_content"
         android:id="@+id/editText"
         android:layout_width="match_parent"/>

      <TableLayout
         android:layout_height="wrap_content"
         android:layout_width="wrap_content"
         android:layout_gravity="center"
         android:stretchColumns="*">

         <TableRow>

            <Button
               android:id="@+id/button1"
               android:layout_height="wrap_content"
               android:layout_width="match_parent"
               android:text="Первая  кнопка"/>

            <Button
               android:id="@+id/button2"
```

```
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="Вторая кнопка"/>
    </TableRow>

    <TableRow>

        <Button
            android:id="@+id/button3"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            android:text="Третья кнопка"/>

        <Button
            android:id="@+id/button4"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            android:text="Четвёртая кнопка"/>
    </TableRow>

    <TableRow>

        <Button
            android:id="@+id/button5"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            android:text="Пятая кнопка"/>

        <Button
            android:id="@+id/button6"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            android:text="Шестая кнопка"/>
    </TableRow>
</TableLayout>
```

```
    </LinearLayout>
</LinearLayout>
```

Запускаем приложение и проверяем. Теперь при повороте экрана видны все кнопки (рисунок Л.3.5).

Рисунок Л.3.5 – Внешний вид приложения в альбомном режиме (верная компоновка)

При создании альтернативной разметки не забывайте включать все компоненты, к которым будете обращаться программно, иначе возникнет ошибка. Допустим, вы забыли добавить шестую кнопку. В портретном режиме программа будет работать, а когда пользователь перевернёт экран, то активность будет инициализировать все компоненты для работы, а кнопки-то и нет.

Существует еще один быстрый способ создания нового файла для альбомной ориентации без участия мастера настройки.

Переключитесь в режим дизайна основной разметки. Наверху на панели инструментов есть значок листа с логотипом Android (самый первый значок). Щёлкните на нём и выберите пункт **Create Landscape Variation**. Появится готовый файл в папке **res/layout-land**. Содержимое файла из портретной ориен-

тации будет скопировано в файл, и его можно отредактировать по своему желанию.

3.2 Программное определение ориентации

Чтобы из кода узнать текущую ориентацию, можно создать следующую функцию:

```
private String getScreenOrientation(){
   if (getResources().getConfiguration().orientation ==
                Configuration.ORIENTATION_PORTRAIT)
      return "Портретная ориентация";
   else
   if (getResources().getConfiguration().orientation ==
                Configuration.ORIENTATION_LANDSCAPE)
          return "Альбомная ориентация";
   else
          return "";
}
```

Вызовите данную функцию из нужного места, например, при щелчке кнопки и узнайте текущую ориентацию. В примере использованы две распространённые системные константы для ориентации. Существует также константа **ORIENTATION_SQUARE** (квадратный экран). Но такие телефоны практически не встречаются.

Можно также вычислить ширину и высоту экрана. Если высота больше ширины, то устройство находится в портретной ориентации, иначе – в альбомной:

```
private String mOrientation = "";

private boolean isLandscapeMode(Activity activity)
{
   int width = activity.getWindowManager().
```

```
            getDefaultDisplay().getWidth();
    int height = activity.getWindowManager().
            getDefaultDisplay().getHeight();

    boolean isLandscape = width > height;

    if (isLandscape)
        mOrientation = "Альбомная";
    else
        mOrientation = "Портретная";

    return isLandscape;
}
```

В настоящее время такой код считается устаревшим, и для вычисления размера экрана используются другие методы.

3.3 Программное определение поворота устройства

Мы научились определять текущую ориентацию, но как понять, в какую сторону повернули устройство? Ведь его можно повернуть влево, вправо или вообще перевернуть. Для этого можно использовать следующую функцию:

```
private String getRotateOrientation() {
    int rotate = getWindowManager().getDefaultDisplay().
            getRotation();
    switch (rotate) {
    case Surface.ROTATION_0:
        return "Не поворачивали";
    case Surface.ROTATION_90:
        return "Повернули на 90 градусов по часовой
                стрелке";
    case Surface.ROTATION_180:
        return "Повернули на 180 градусов";
```

```
    case Surface.ROTATION_270:
            return "Повернули  на 90 градусов против часовой
                    стрелки";
    default:
            return "Не понятно";
    }
}
```

Раньше существовал аналогичный метод **getOrientation()**, который устарел. В настоящее время используется **getRotation().**

3.4 Программная установка ориентации

Если вдруг для каких-то целей возникла необходимость запустить приложение в стиле «вид сбоку», то это можно сделать программно. Разместите код в методе **onCreate()**:

```
import android.content.pm.ActivityInfo;

setRequestedOrientation(ActivityInfo.
            SCREEN_ORIENTATION_LANDSCAPE);
```

Можно также запретить приложению менять ориентацию, если добавить нужный код в **onCreate()**:

```
//для альбомного режима
setRequestedOrientation  (ActivityInfo.
                SCREEN_ORIENTATION_LANDSCAPE);
// или для портретного режима
setRequestedOrientation  (ActivityInfo.
                SCREEN_ORIENTATION_PORTRAIT);
```

Но указанный способ не совсем желателен. Лучше установить нужную ориентацию через манифест, прописав в элементе <activity> параметр **android:screenOrientation**:

android:screenOrientation="portrait"
android:screenOrientation="landscape"

Кстати, существует ещё один вариант, когда устройство полагается на показания сенсора и некоторые другие:
android:screenOrientation="sensor"

3.5 Запрет на создание новой активности

На примере программной установки ориентации можно увидеть интересный эффект, о котором нужно помнить. Предположим, в приложении есть кнопка, позволяющая менять ориентацию. Заодно будем менять текст на кнопке, чтобы операция соответствовала надписи.

```
public class MainActivity extends Activity {

    private Button mButton;
    static final String ORIENTATION_PORTRAIT =
            "Портретный режим";
    static final String ORIENTATION_LANDSCAPE =
            "Альбомный режим";

    // определяем изменение ориентации экрана
    boolean mState = false;

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

            mButton = (Button) findViewById(R.id.button);

            // установим текст по умолчанию
            mButton.setText(ORIENTATION_LANDSCAPE);
```

```
    }

    public void onClick(View view) {
        // state FALSE: переключаемся на LANDSCAPE
        if (!mState) {
            setRequestedOrientation(ActivityInfo.
              SCREEN_ORIENTATION_LANDSCAPE);
            mButton.setText
              (ORIENTATION_PORTRAIT);
        }
        // state TRUE: переключаемся на PORTRAIT
        else {
            setRequestedOrientation(ActivityInfo.
              SCREEN_ORIENTATION_PORTRAIT);
            mButton.setText
              (ORIENTATION_LANDSCAPE);
        }
        // обновляем state на противоположное значение
        mState = !mState;
    }
}
```

Запустите проект и нажмите на кнопку. Ориентация экрана поменялась, однако текст на кнопке остался прежним, хотя по идее он должен был измениться (рисунок Л.3.6).

Нажмём на кнопку ещё раз. Надпись изменится, но ориентация не сменится. И только повторный щелчок повернёт экран в обратную сторону.

По умолчанию, при смене ориентации Android уничтожает и пересоздаёт активность из кода, что подразумевает повторный вызов метода **onCreate()**. Поэтому при повороте активность устанавливала текст, определенный в **onCreate()**. В большинстве случаев это не мешает программе. Но если приложение воспроизводит видео, то при смене ориентации вызов

onCreate() может привести к повторному началу воспроизведения.

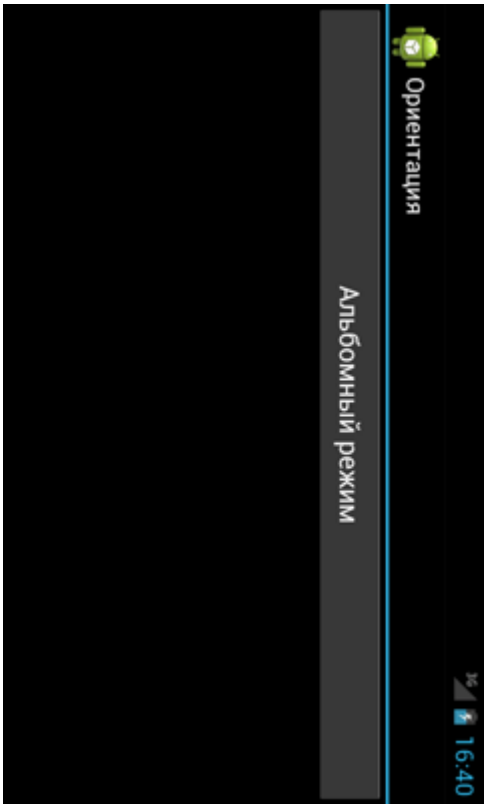

Рисунок Л.3.6 – Неверный результат работы приложения

Чтобы активность не пересоздавалась, добавьте в манифест строчку для нужной активности:

```
android:configChanges="keyboardHidden|orientation|screenSize"
```

В этом случае система вызовет метод **onConfigurationChanged(Configuration)** и полагается на разработчика:

```
@Override
public void onConfigurationChanged
        (Configuration newConfig) {
   super.onConfigurationChanged(newConfig);
   // Проверяем ориентацию экрана
   if (newConfig.orientation == Configuration.
```

```
                    ORIENTATION_LANDSCAPE) {
        Toast.makeText(this, "landscape",
                    Toast.LENGTH_SHORT).show();
    }
    else if (newConfig.orientation == Configuration.
                    ORIENTATION_PORTRAIT) {
        Toast.makeText(this, "portrait",
                    Toast.LENGTH_SHORT).show();
    }
}
```

3.6 Проблема «исчезающего текста»

Как уже говорилось, при смене ориентации активность пересоздаётся. При этом можно наблюдать интересный эффект с пропадающим текстом. Для примера, создадим два текстовых поля. Одному из них присвоим идентификатор, а другое поле оставим без него.

```
<EditText
    android:id="@+id/editTest"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Запустите приложение, введите любой текст в обоих полях и смените ориентацию. Вы увидите, что у поля с идентификатором текст при повороте сохранится, а у поля без идентификатора текст исчезнет. Учитывайте данное обстоятельство.

К вышесказанному можно добавить, что при смене ориентации у поля с идентификатором вызывается метод **onTextChanged()**:

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

    EditText editText = (EditText)findViewById
                        (R.id.editTextTest);
    editText.addTextChangedListener(new TextWatcher() {
      @Override
      public void onTextChanged(CharSequence s, int start, int
before, int count) {
            Toast.makeText(MainActivity.this,
                    "onTextChanged: " + s,
                   Toast.LENGTH_SHORT).show();
      }

      @Override
      public void beforeTextChanged(CharSequence s, int start,
                int count, int after) {

      }

      @Override
      public void afterTextChanged(Editable s) {

      }
    });
}
```

Если используются две разные разметки, то возможна ситуация, когда в альбомной ориентации используется кнопка, которой нет в портретной ориентации. Это может привести к ошибке в коде, поэтому нужно проверить существование кнопки:

```
Button landscapeButton = (Button) findViewById
                         (R.id.landscapeButton);
if (landscapeButton != null) {
   // Можно работать
}
```

На практике такое встречается редко, но, тем не менее, данный момент следует учитывать.

3.7 Запоминание значений переменных

С поворотом экрана повляется еще одна очень неприятная проблема,возникающая из-за того, что при повороте экрана активность создаётся заново.

Пусть у нас имеется приложение, в котором на экране выводится значение некоторого счетчика **mCount**. Допустим, на экране написано "Значение счетчика равно 5". Поворачиваем экран – и надпись меняется на "Значение счетчика равно 0".

Происходит это за счет того, что активность при повороте создаётся заново. А значит, переменная **mCount** снова принимает значение 0, т. е. сбрасывается в начальное значение.

Что же делать? Для этих целей у активности существует специальный метод **onSaveInstanceState()**, который вызывается системой перед методами **onPause()**, **onStop()** и **onDestroy()**. Метод позволяет сохранить значения простых типов в объекте **Bundle**. Класс **Bundle** – это простой способ хранения данных ключ/значение.

Создадим ключ с именем **KEY_COUNT**. В Android Studio с версии 1.5 появились живые шаблоны, позволяющие быстро создать ключ. Введите до метода **onCreate()** строчными буквами слово **key**, во время набора появится подсказка. Нажимаем **Enter** и получаем заготовку. После символа подчёркивания вводим название ключа. В результате получим ключ следующего вида:

```
private static final String KEY_COUNT = "COUNT";
```

Далее создаём метод **onSaveInstanceState()** после метода **onCreate()**. Во время набора имени метода подсказка покажет, что имеется два метода. Выбирайте метод с одним параметром (обычно он идёт вторым). Записываем в ключ значение счётчика:

```
@Override
protected void onSaveInstanceState(Bundle outState) {
   super.onSaveInstanceState(outState);

   outState.putInt(KEY_COUNT, mCount);
}
```

А в методе **onCreate()** делаем небольшую проверку.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);

   setContentView(R.layout.activity_main);

   mInfoTextView = (TextView) findViewById(R.id.textView);

   if (savedInstanceState != null) {
      mCount = savedInstanceState.getInt(KEY_COUNT, 0);
      mInfoTextView.setText("Значение счетчика равно " +
                        mCount);
   }
}
```

У метода в параметре содержится объект **Bundle**. Только здесь он назван **savedInstanceState** вместо **outState**, но это не имеет значения. Имена можно придумывать самим. Главное, что объект содержит сохранённое значение переменной при повороте. При первом запуске приложения объект не существует

(**null**), а потом мы его создали своим кодом. Для этого и нужна проверка. Обратите внимание, что здесь мы не прибавляем единицу к счётчику, как у кнопки. Если скопировать код у кнопки, то получится, что счётчик будет увеличиваться самостоятельно при поворотах без нажатия на кнопку.

Обратите внимание, что данный способ используется для сохранения промежуточных результатов во время действия программы.

3.8 Ориентация у фрагментов

Может возникнуть ситуация, когда нужно выводить конкретный фрагмент в нужной ориентации. У фрагментов есть собственный жизненный цикл, и в методах фрагмента можно реализовать свой код:

```
@Override
public void onResume() {
   getActivity().setRequestedOrientation(ActivityInfo.
                  SCREEN_ORIENTATION_FULL_SENSOR);
}

@Override
public void onPause() {
   getActivity().setRequestedOrientation(ActivityInfo.
                  SCREEN_ORIENTATION_PORTRAIT);
   // ваш код
   super.onPause();
}
```

Таким образом, если рассмотреть жизненный цикл активности при смене ориентации, можно сказать, что при повороте активность проходит через цепочку различных состояний. Порядок следующий:

1. onPause()

2. onStop()
3. onDestroy()
4. onCreate()
5. onStart()
6. onResume()

4 Контрольные вопросы

1. Перечислите известные вам способы создания альтернативной разметки для смены ориентации устройства.
2. Как программно определить текущую ориентацию экрана устройства?
3. Как программно определить, на сколько градусов и в какую сторону было повернуто устройство?
4. Как программно установить ориентацию устройства?
5. Как запретить повторное создание активности при смене ориентации устройства?
6. Каким образом можно сохранить значения переменных при смене ориентации устройства?
7. Через какие состояния проходит активность при повороте экрана?

5 Задание для самостоятельного выполнения

Разработать приложение для работы с матрицами произвольного размера согласно варианту. Предусмотреть возможность ручного ввода элементов матрицы, а также заполнения их случайными числами. Обеспечить различные виды разметки пользовательского интерфейса в зависимости от ориентации экрана. При повороте экрана введенные пользователем данные и результаты расчета должны сохраняться.

Варианты заданий

1. Дан двумерный массив, заполненный целыми числами. Сформировать одномерный массив, каждый элемент которого равен количеству элементов соответствующей строки, больших

заданного числа.

2. Определить, есть ли в данной матрице строка, состоящая только из элементов, принадлежащих промежутку от A до B. A и B вводятся с клавиатуры.

3. Дан двумерный массив, заполненный целыми числами. Сформировать одномерный массив, каждый элемент которого равен первому четному элементу соответствующего столбца, если такого нет, то равен нулю.

4. Дана матрица. Поменять местами первую строку и строчку, в которой находится первый нулевой элемент.

5. Найти сумму двух матриц размером $m \times n$.

6. Дан двумерный массив размером $m \times n$. Определить, есть ли в данном массиве столбец, в котором равное количество положительных и отрицательных элементов.

7. Дана матрица A размерностью $m \times n$. Сформировать одномерный массив B, элементами которого являются номера первых отрицательных элементов каждой строки массива A. (0 - отрицательный элемент отсутствует).

8. Дан двумерный массив, заполненный целыми числами. Сформировать одномерный массив, каждый элемент которого равен наибольшему по модулю элементу соответствующего столбца.

9. Дана матрица. Найти количество элементов в каждой строке, больших среднего арифметического элементов данной строки.

10. Найти среднее арифметическое элементов каждой строки матрицы Q(m,n) и вычесть его из элементов этой строки.

11. Дан двумерный массив размером $m \times n$, заполненный случайными числами. Определить, есть ли в данном массиве строка, содержащая больше положительных элементов, чем отрицательных.

12. Дана матрица K(m,n). Сформировать одномерный массив L(m), элементами которого являются суммы элементов j-ого столбца, принадлежащих диапазону от a до b.

13.Матрица K(m,n) состоит из нулей и единиц. Найти в ней номера строк, содержащих равное количество нулей и единиц, либо сообщить, что таких нет.

14.Дана матрица K(m,n). Найти максимальный элемент в каждой строке и заменить его заданным числом.

15.Для заданной матрицы A(m,n) найти ее норму: $\|A\| = \max\limits_{i=1,m} \sum\limits_{k=1}^{n} |a_{ik}|$.

16.Целочисленный массив K(m,n) заполнить нулями и единицами, расположив их в шахматном порядке.

17.Дана матрица A(m,n). Сформировать одномерный массив B(m), элементами которого являются суммы элементов i-ой строки, начиная с первого отрицательного элемента и до конца строки матрицы (0 – если отрицательный элемент отсутствует).

18.Дан двумерный массив, заполненный целыми числами. Сформировать одномерный массив, каждый элемент которого равен произведению четных положительных элементов соответствующего столбца.

19.Дан двумерный массив. Заменить все элементы первых трех столбцов на их квадраты, в остальных столбцах изменить знак каждого элемента на противоположный.

20.Дана матрица. Поменять местами две средние строки с первой и последней.

21.Дан двумерный массив. Заменить максимальный элемент каждой строки на противоположный по знаку.

22.Дана матрица. Сформировать вектор, элементами которого могут быть 1 или 0 в зависимости от следующего условия: 1 – сумма элементов строки больше вводимого числа X, 0 – иначе.

23.Определить, есть ли в данном массиве строка, состоящая только из отрицательных элементов.

24.Дана матрица. Поменять местами первый и последний столбцы.

25.Дан двумерный массив, заполненный целыми числами. Сформировать одномерный массив, каждый элемент которого

равен количеству отрицательных элементов, кратных 3 или 5, соответствующей строки.

26.В каждой строке заполненной случайным образом матрицы размером $m \times n$ поменять местами первый элемент и максимальный по модулю.

27.Дан двумерный массив. Поменять местами средние строки.

28.В матрице Z(n,n) найти суммы элементов главной и побочной диагоналей.

29.Дан двумерный массив размером $m \times n$. Определить, есть ли в данном массиве строка, в которой ровно два отрицательных элемента.

30.В матрице Z(n,n) каждый элемент разделить на диагональный, стоящий в том же столбце.

31.Определить, есть ли в данном массиве столбец, состоящий только из положительных или нулевых элементов.

32.Элементы одномерного массива $A(n^2)$ построчно расположить в матрице B(n,n).

Лабораторная работа №4. Работа с уведомлениями

1 Цель работы

Цель работы – ознакомиться и получить практические навыки создания и настройки уведомлений при разработке мобильных приложений.

2 Содержание отчета

- титульный лист;
- краткое теоретическое описание;
- задание на лабораторную работу, включающее математическую формулировку задачи;
- результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений;

3 Порядок выполнения работы

Кроме Toast-уведомлений, существует также другой тип уведомлений, который выводится в системной строке состояния в виде значка с небольшим текстом. Если открыть окно уведомлений, то можно увидеть расширенную текстовую информацию об уведомлении.

Когда пользователь открывает расширенное сообщение, Android запускает объект **Intent**, который определён в соответствии с уведомлением. Можно также конфигурировать уведомление с добавлением звука, вибрации и мигающих индикаторов на мобильном устройстве.

Этот вид уведомления удобен в том случае, когда приложение работает в фоновом режиме и должно уведомить пользователя о каком-либо важном событии. Уведомление будет висеть до тех пор, пока пользователь не отреагирует на него, в отличие от Toast-сообщения, которое исчезнет через несколько секунд. Фоновое приложение создаёт уведомление в строке состояния, но не запускает активность самостоятельно для получения

пользовательского взаимодействия. Это должен сделать только сам пользователь в удобное ему время.

3.1 Создание простых уведомлений

Чтобы создать уведомление в строке состояния, необходимо использовать два класса:

- **Notification** – определяет свойства уведомления строки состояния: значок, расширенное сообщение и дополнительные параметры настройки (звук и др.);
- **NotificationManager** – системный сервис Android, который управляет всеми уведомлениями. Экземпляр класса **NotificationManager** создается при помощи вызова метода **getSystemService()**, а затем, когда надо показать уведомление пользователю, вызывается метод **notify()**. Существует также более простой способ его использования через метод **from()**.

Начиная с Android 3, для уведомлений используется класс **Notification.Builder**. Добавим на экран активности кнопку и создадим простой пример для демонстрации работы уведомления.

```
package ru.username.testapplication;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.content.res.Resources;
import android.graphics.BitmapFactory;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
```

```
import android.view.View;

public class MainActivity extends ActionBarActivity {
   // Идентификатор уведомления
   private static final int NOTIFY_ID = 101;

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

   public void onClick(View view) {
      Context context = getApplicationContext();

      Intent notificationIntent = new Intent(context, MainActivi-
ty.class);
      PendingIntent contentIntent = PendingIntent.getActivity
            (context, 0, notificationIntent,
            PendingIntent.FLAG_CANCEL_CURRENT);

      Resources res = context.getResources();
      Notification.Builder builder = new Notification.Builder
            (context);

      builder.setContentIntent(contentIntent)
         .setSmallIcon(R.drawable.ic_launcher_cat)
         // большая картинка
         .setLargeIcon(BitmapFactory.decodeResource
               (res, R.drawable.hungrycat))
         // текст в строке состояния
          //.setTicker(res.getString(R.string.warning))
         .setTicker("Последнее китайское
               предупреждение!")
         .setWhen(System.currentTimeMillis())
```

```
                .setAutoCancel(true)
                // Заголовок уведомления
                //.setContentTitle(res.getString(R.string.notifytitle))
                .setContentTitle("Напоминание")
                //.setContentText(res.getString(R.string.notifytext))
                // Текст уведомления
                .setContentText("Пора покормить кота");

        // до API 16
        // Notification notification = builder.getNotification();
        Notification notification = builder.build();

        NotificationManager notificationManager =
                (NotificationManager) context.getSystemService
                (Context.NOTIFICATION_SERVICE);
        notificationManager.notify(NOTIFY_ID, notification);
    }
}
```

Для начала необходимо создать идентификатор уведомления. Он нужен, чтобы можно было различать уведомления друг от друга. Если будет один идентификатор, то каждое новое уведомление затрёт предыдущее. Для идентификатора используется какое-либо число.

Далее формируется внешний вид и поведение уведомления через построитель **Notification.Builder**. Здесь можно задать текст уведомлений, значки и прочие атрибуты:

- метод **setSmallIcon()** устанавливает маленький значок, который выводится в строке состояния, а также в правой части открытого уведомления;
- метод **setLargeIcon()** устанавливает большой значок, который выводится в открытом уведомлении слева;
- метод **setTicker()** выводит временную строку в строке состояния, которая затем исчезает. Остаётся только маленький значок.

Остальные методы понятны по названиям. «Правильные варианты» использования строк через ресурсы закомментированы, но для лучшего понимания в примерах сразу показаны нужные строки.

Также нам необходимы объекты **Intent** и **PendingIntent**, которые описывают намерения и целевые действия. В нашем случае мы хотим запустить активность, когда пользователь среагирует на уведомление.

Начиная с API 16 вместо устаревшего метода **getNotification()** следует использовать метод **build()**.

Далее нужно сформировать уведомление с помощью специального менеджера. Ссылку на **NotificationManager** можно получить через вызов метода **getSystemService()**, передав ему в качестве параметра строковую константу **NOTIFICATION_SERVICE**, определённую в классе **Context.**

Выводится уведомление с помощью метода **notify()** – своеобразный аналог метода **show()** у класса **Toast**.

При запуске приложения прежде всего мы увидим, как появился маленький значок. Также появляется текст "Последнее китайское предупреждение!", который быстро исчезает (рисунок Л.4.1).

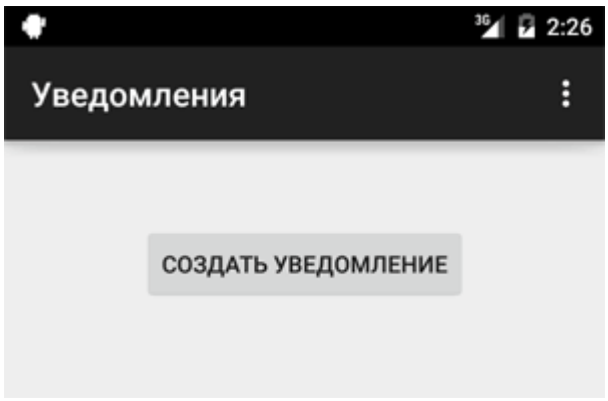

Рисунок Л.4.1 – Создание обычного уведомления

При проверке кода под Android 5.0 можно увидеть, что маленький значок из метода **setSmallIcon()** стал белым (как на скриншоте), а метод **setTicker()** не срабатывает. Значок выгля-

дит таким образом из-за того, что в Android 5.0 используется дизайн Material, который требует использовать для значков в уведомлениях белый цвет на прозрачной подложке. Учитывайте это обстоятельство и заранее подготавливайте такой ресурс. Вообще, в Android 5.0 уведомления сильно переработаны, и их можно выводить даже на экран блокировки и на часы.

Далее мы можем открыть уведомление, чтобы увидеть более подробную информацию (рисунок Л.4.2).

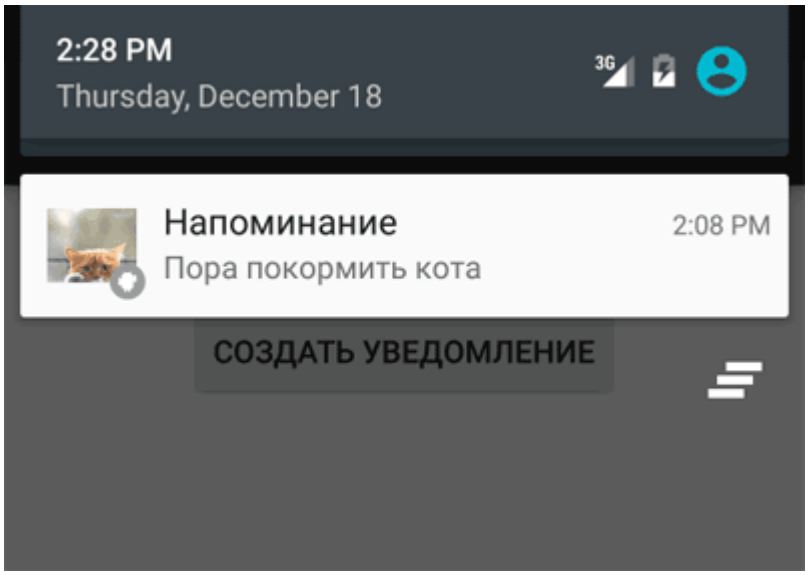

Рисунок Л.4.2 – Расширенное уведомление

В примере показан избыточный код, чтобы показать больше возможностей уведомлений. Часть методов с префиксом **set** можно пропустить. Ниже показан укороченный код с использованием библиотеки совместимости для старых устройств. Обратите также внимание на упрощённый вариант доступа к менеджеру уведомлений через метод **NotificationManagerCompat.from(this)**.

```
Notification.Builder builder = new NotificationCompat.Builder
                    (this);
// оставим только самое необходимое
builder.setContentIntent(contentIntent)
```

```
        .setSmallIcon(R.drawable.ic_launcher_cat)
        .setContentTitle("Напоминание")
        // Текст уведомления
        .setContentText("Пора  покормить кота");

    Notification notification = builder.build();

    NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);
    notificationManager.notify(NOTIFY_ID,  notification);
```

Как уже упоминалось, если нужно обновить уведомление, то просто ещё раз отправьте его устройству под этим же идентификатором.

Если коснуться уведомления, то запустится наше приложение (даже если оно было перед этим закрыта).

Совсем не обязательно запускать своё приложение, хотя это является распространённой практикой. Можно задать нужное поведение, например, запустить сайт по указанному адресу. Переделаем код:

```
    Context context = getApplicationContext();
    Intent notificationIntent = new Intent(Intent.ACTION_VIEW,
        Uri.parse("http://kubsau.ru"));
    PendingIntent  pendingIntent  = PendingIntent.getActivity
                (context,  0, notificationIntent,
                PendingIntent.FLAG_CANCEL_CURRENT);

    Notification.Builder builder = new Notification.Builder(context)
        .setContentTitle("Посетите  сайт")
        .setContentText("http://  kubsau.ru ")

.setTicker("Внимание!").setWhen(System.currentTimeMillis())
        .setContentIntent(pendingIntent)
```

```
.setDefaults(Notification.DEFAULT_SOUND).setAutoCancel(true)
        .setSmallIcon(R.drawable.ic_launcher);

    NotificationManager notificationManager = (NotificationMan-
ager) getSystemService(Context.NOTIFICATION_SERVICE);
    notificationManager.notify(NOTIFY_ID, builder.build());
```

Обратите внимание, что на этот раз мы не указали картинку для большого значка и система подставляет в этом случае маленький значок, растягивая его до нужных размеров. Также появился новый метод **setDefaults()**, о котором говорится ниже.

Также можно вывести индикатор прогресса, чтобы указать текущий ход выполнения задачи. Можно установить бесконечное выполнение:

```
setProgress(100, 50, false);
```

Если цель – только вывести уведомление, но не запускать активность при нажатии на самом уведомлении, то используйте вызов намерения без параметров:

```
Intent intent = new Intent();
```

Можно удалить из программы своё уведомление.

```
notificationManager.cancel(NOTIFY_ID);
```

3.2 Настройка уведомлений

В уведомления можно добавлять вибрацию, звуковой сигнал или мерцание светодиодами при помощи настроек по умолчанию. В свойстве **defaults** допускается сочетание следующих констант:

- Notification.DEFAULT_LIGHTS;
- Notification.DEFAULT_SOUND;
- Notification.DEFAULT_VIBRATE.

Чтобы к уведомлению добавить звук и вибрации по умолчанию, используется код:

```
notification.defaults = Notification.DEFAULT_SOUND |
```

Notification.DEFAULT_VIBRATE;

Если требуется установить сразу все значения по умолчанию, задействуется константа **Notification.DEFAULT_ALL**.

Использование звуковых оповещений для уведомления пользователя о событиях, связанных с устройством (например, входящий звонок), стало привычным. Большинство стандартных событий, от входящих звонков до новых сообщений и низкого заряда батареи, объявляются с помощью звуковых мелодий. Android позволяет проигрывать любой звуковой файл на телефоне в качестве уведомления. Чтобы это сделать, нужно присвоить свойству **sound** путь **URI**:

notification.sound = ringURI;

Также можно использовать собственный звуковой файл, загруженный на устройстве или добавленный в проект в качестве ресурса:

```
Uri ringURI =
    RingtoneManag-
er.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
notification.sound = ringURI;
```

Если файл находится на SD-карте, то путь к нему можно задать следующим образом:

notification.sound = Uri.parse("file:///sdcard/cat.mp3");

Можно также использовать функцию виброзвонка в телефоне, чтобы сопровождать уведомление вибрацией для привлечения внимания пользователя.

Чтобы использовать виброзвонок, передайте в свойство **vibrate** объекта **Notification** массив значений типа **long**. Постройте массив, учитывая, что значения, отвечающие за продолжительность вибрации (в миллисекундах), чередуются со значениями, которые означают длину паузы между вибрациями.

Прежде чем использовать виброзвонок в своем приложении, необходимо получить нужные полномочия, прописав их в манифесте:

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

В следующем примере показано, как изменить уведомление, чтобы одна секунда вибрации сменялась одной секундой паузы на протяжении пяти секунд:

```
long[] vibrate = new long[] { 1000, 1000, 1000, 1000, 1000 };
notification.vibrate = vibrate;
```

В настоящее время эмулятор Android не умеет оповещать о вибрации ни визуально, ни с помощью звуковых сигналов.

Объект **Notification** включает в себя свойства для настройки цвета и частоты мерцания светодиодов устройства. Здесь стоит обратить внимание, что конкретные модели устройств могут не содержать светодиодные индикаторы или иметь другие цвета.

Свойство **ledARGB** может устанавливать цвет для светодиодной подсветки. Свойства **ledOffMS** и **ledOnMS** позволяют регулировать частоту и поведение светодиодов. Вы можете включить светодиоды, присвоив свойству **ledOnMS** значение 1, а **ledOffMS** – 0. Присвоив им обоим значения 0, светодиоды можно выключить.

Настроив работу со светодиодами, необходимо также добавить флаг **FLAG_SHOW_LIGHTS** к свойству **flags** объекта **Notification**.

В следующем фрагменте кода показано, как включить на устройстве красный светодиод:

```
notification.ledARGB  = Color.RED;
notification.ledOffMS  = 0;
notification.ledOnMS  = 1;
notification.flags  = notification.flags |
            Notification.FLAG_SHOW_LIGHTS;
```

В настоящее время эмулятор Android не умеет визуально показывать активность светодиодов.

Уведомления можно делать текущими и/или настойчивыми, устанавливая флаги **FLAG_INSISTENT** и **FLAG_ONGOING_EVENT**. Уведомления, помеченные как текущие, используются для представления событий, которые выполняются в данный момент времени (например, загрузка файла, фоновое проигрывание музыки). Текущие уведомления необходимы для сервисов, работающих на переднем плане. Пример установки флагов:

```
notification.flags = notification.flags |
                    Notification.FLAG_ONGOING_EVENT;
```

В расширенной статусной строке текущие события отделены от обычных, чтобы их можно было сразу отличить.

Настойчивые уведомления непрерывно повторяют звуковые сигналы, вибрируют и мерцают светодиодами, пока не будут остановлены. Подобные уведомления, как правило, используются для событий, которые требуют немедленного и своевременного внимания, таких как входящий звонок или срабатывание будильника. В следующем фрагменте кода показано, как сделать уведомление настойчивым:

```
notification.flags = notification.flags |
                    Notification.FLAG_INSISTENT;
```

В методе **getActivity()** может понадобиться изменить флаг, например:

```
PendingIntent pendingIntent = PendingIntent.getActivity
 (context, 0, intent, Intent.FLAG_ACTIVITY_NEW_TASK);
```

Существуют и другие флаги. Хотя в большинстве случаев используется просто 0.

Начиная с Android 5.0 пользователь может установить собственный уровень оповещений, нажав на кнопки увеличения громкости на домашнем экране. Появится диалоговое окно, в котором задаётся один из трёх доступных уровней (рисунок Л.4.3).

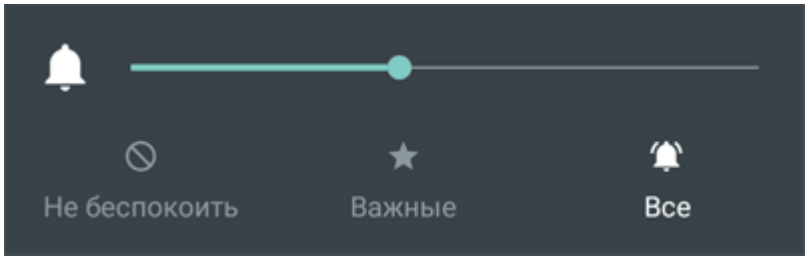

Рисунок Л.4.3 – Настройка громкости оповещений

Не сразу бывает заметно, но на самом деле, когда при нажатии на уведомлении запускается активность, то запускается не старая активность, которая была на экране до этого, а новая. Это можно увидеть в примере, в котором, например, есть текстовое поле с текстом. Введите какой-нибудь текст в активности, а потом создайте уведомление, вызывающее активность. Вы увидите, что запустится новая активность с пустым текстовым полем, хотя мы ожидали увидеть запущенную активность. Если вам нужен именно этот вариант, то используйте флаги для намерения.

```
Intent intent = new Intent(context,  MainActivity.class);

intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
            | Intent.FLAG_ACTIVITY_SINGLE_TOP);
```

Либо можно прописать в манифесте для нужной активности атрибут **android:launchMode="singleTop"**.

3.3 Анимированный значок для уведомления

Рассмотрим следующую ситуацию на примере кода одного из ранее рассмотренных приложений. Заменим в нем одну строчку, которая отвечает за вывод маленького значка - **.setSmallIcon(android.R.drawable.stat_sys_upload)**:

```
public void onClick(View view) {
    Context context = getApplicationContext();
```

```
        Intent notificationIntent = new Intent(context,
                TestActivity.class);
        PendingIntent contentIntent = PendingIntent.getActivity
                (context, 0 , notificationIntent,
                PendingIntent.FLAG_CANCEL_CURRENT);

        NotificationManager nm = (NotificationManager) context.
              getSystemService
              (Context.NOTIFICATION_SERVICE);

        Resources res = context.getResources();
        Notification.Builder builder = new Notification.Builder
              (context);

        builder.setContentIntent(contentIntent)
                  .setSmallIcon(android.R.drawable.stat_sys_upload)
                  .setLargeIcon(BitmapFactory.decodeResource
                     (res, R.drawable.cat))
                  .setTicker("Последнее предупреждение!")
                  .setWhen(System.currentTimeMillis())
                  .setAutoCancel(true)
                  .setContentTitle("Напоминание")
                  // Текст уведомления
                  .setContentText("Пора покормить кота");

        Notification n = builder.getNotification();

        nm.notify(NOTIFY_ID, n);
    }
```

Запускаем код и создаём уведомление. Вы увидите, что в строке состояния выводится анимированный значок стрелки (рисунок Л.4.4). Такой способ стоит использовать для действительно важных сообщений, чтобы понапрасну не отвлекать пользователя.

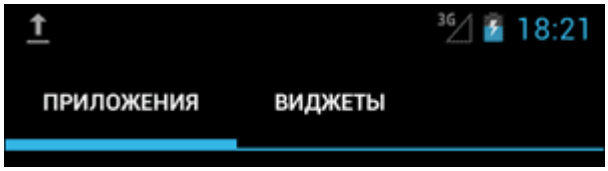

Рисунок Л.4.4 – Анимированный значок уведомления

С анимированным значком связана интересная особенность. Если вы опустите метод **setTicker()**, то значок уже не будет анимированным.

Можно попробовать использовать другие системные анимации, например, **android.R.drawable.stat_sys_download** или создать собственную анимацию.

3.4 Дополнительные возможности уведомлений

Начиная с Android 4.1-4.2 появились дополнительные возможности для уведомлений. Рассмотрим их на примере.

Подготовим разметку из четырёх кнопок (рисунок Л.4.5):

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
      android:id="@+id/btBasicNotification"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:onClick="sendActionNotification"
      android:text="Уведомление с кнопками" />

    <Button
      android:id="@+id/btBigTextNotification"
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="sendBigTextStyleNotification"
        android:text="Уведомление с длинным текстом" />

    <Button
        android:id="@+id/btBigPictureNotification"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="sendBigPictureStyleNotification"
        android:text="Уведомление с большой картинкой" />

    <Button
        android:id="@+id/btInboxStyleNotification"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="sendInboxStyleNotification"
        android:text="Уведомление в стиле Inbox" />

</LinearLayout>
```

Перейдём к коду. У каждой кнопки прописан свой обработчик касания. Поэтому будет удобно разбить код по методам. Обратите внимание, как теперь следует получить доступ к объекту **NotificationManager** – метод **getNotification()** в новых версиях считается устаревшим.

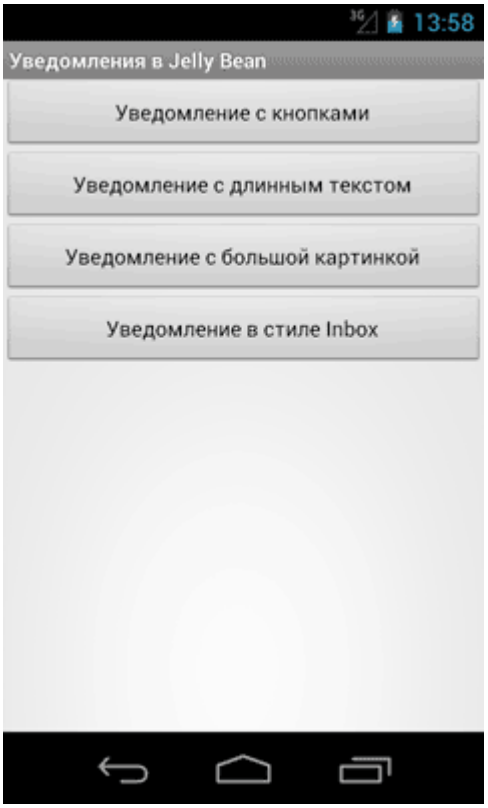

Рисунок Л.4.5 – Дополнительные возможности уведомлений

Начнём с первого варианта. В уведомлениях можно размещать до трёх кнопок. Это может быть удобным, если приложение состоит из нескольких активностей или нужно предложить три разных варианта развития сценария. За появление кнопок в уведомлении отвечает метод **setAction()**.

```
public void sendActionNotification(View view) {
    NotificationManager notificationManager =
            (NotificationManager) getSystemService
            (NOTIFICATION_SERVICE);
    // Намерение для запуска второй активности
    Intent intent = new Intent(this, SecondActivity.class);
    PendingIntent pIntent = PendingIntent.getActivity
            (this, 0, intent, 0);

    // Строим уведомление
```

```
Notification builder = new Notification.Builder(this)
                .setTicker("Пришла посылка!")
                .setContentTitle("Посылка")
                .setContentText("Это я, почтальон
                        Печкин. Принес для вас посылку")

.setSmallIcon(R.drawable.ic_launcher).setContentIntent
                        (pIntent)
                .addAction(R.drawable.ic_launcher,
                        "Открыть", pIntent)
                .addAction(R.drawable.ic_launcher,
                        "Отказаться", pIntent)
                .addAction(R.drawable.ic_launcher,
                        "Другой вариант", pIntent)
                .build();

// убираем уведомление, когда его выбрали
builder.flags |= Notification.FLAG_AUTO_CANCEL;

notificationManager.notify(0, builder);
}
```

Обратите внимание, что у кнопок нет текста, в методе **setAction()** второй параметр служит для удобства разработчика, но пользователь не увидит текст, поэтому следует придумать «говорящие» значки, по которым будет понятен смысл нажатия (рисунок Л.4.6). В нашем примере при нажатии на любой из трёх кнопок запустится вторая активность.

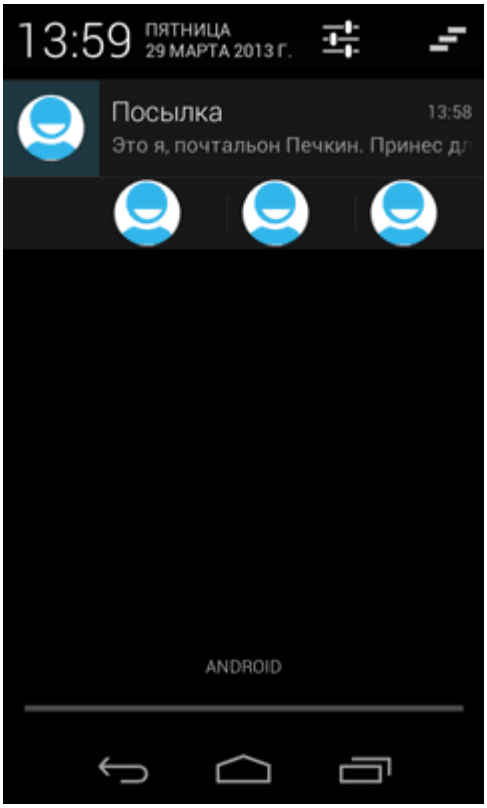

Рисунок Л.4.5 – Уведомление с кнопками

Если текст уведомления слишком длинный, то при использовании метода **setContentText()** он выводится на экран не полностью. Если информация слишком важная и нужно показать ее в уведомлении полностью, то подойдёт следующий вариант:

```
public void sendBigTextStyleNotification(View view) {
    String bigText = "Это я, почтальон Печкин. Принес для
            вас посылку. " + "Только я вам ее не отдам.
            Потому что у вас документов нету. ";

    Intent intent = new Intent(this, SecondActivity.class);
    PendingIntent pIntent = PendingIntent.getActivity
            (this, 0, intent, 0);

    Notification.Builder builder = new Notification.Builder(this)
            .setTicker("Пришла посылка!")
```

```
                .setContentTitle("Уведомление с большим текстом")
                .setContentText("Это я, почтальон Печкин.
                                Принес для вас посылку")
                        .setSmallIcon(R.drawable.ic_launcher)
                        .addAction(R.drawable.ic_launcher,
                                "Запустить активность",
                                pIntent).setAutoCancel(true);

        Notification notification = new Notification.BigTextStyle
                        (builder)
                .bigText(bigText).build();

        NotificationManager notificationManager =
                        (NotificationManager) getSystemService
                        (NOTIFICATION_SERVICE);
        notificationManager.notify(1, notification);
    }
```

Здесь следует обратить внимание на следующий момент. В первом примере переменная **builder** была объектом типа **Notification**. Во втором примере мы разбили построение уведомления на две части. Настройка самого уведомления происходит в объекте типа **Notification.Builder** (сам код остался без изменений), а стиль уведомления задаётся уже для объекта типа **Notification**. В данном случае мы используем стиль **BigTextStyle().bigText()**. В этом случае текст в **setContentText()** игнорируется, а вместо него используется отдельно заданный нами текст в строковой переменной **bigText**.

Пример уведомления с длинным текстом приведен на рисунке Л.4.6.

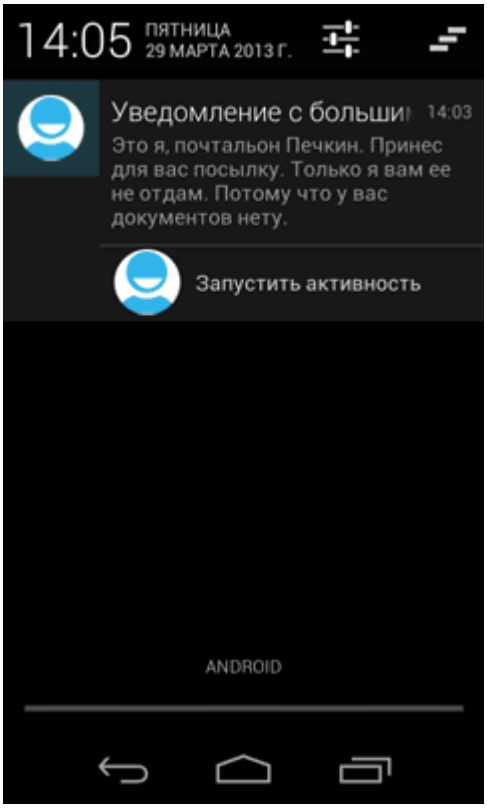

Рисунок Л.4.6 – Уведомление с длинным текстом

Пример с большой картинкой аналогичен с предыдущим примером. Только здесь задается другой стиль для уведомления. Вместо стиля длинного текста используется стиль **BigPictureStyle().bigPicture()**:

```
public void sendBigPictureStyleNotification(View view) {
    Intent intent = new Intent(this, SecondActivity.class);
    PendingIntent pIntent = PendingIntent.getActivity
      (this, 0, intent, 0);
    Builder builder = new Notification.Builder(this);
    builder.setContentTitle("Большая посылка")
      .setTicker("Пришла посылка!")
      .setContentText("Уведомление с большой картинкой")
      .setSmallIcon(R.drawable.ic_launcher)
      .addAction(R.drawable.ic_launcher,
            "Запустить активность", pIntent);
```

```
        // Подготовим большую картинку
        Notification notification = new Notification.BigPictureStyle
                (builder).bigPicture(BitmapFactory.decodeResource
                (getResources(),R.drawable.cat)).build();
        notification.flags |= Notification.FLAG_AUTO_CANCEL;
        NotificationManager notificationManager =
                (NotificationManager) getSystemService
                (NOTIFICATION_SERVICE);
        notificationManager.notify(2, notification);
    }
```

Пример уведомления с большой картинкой показан на рисунке Л.4.7.

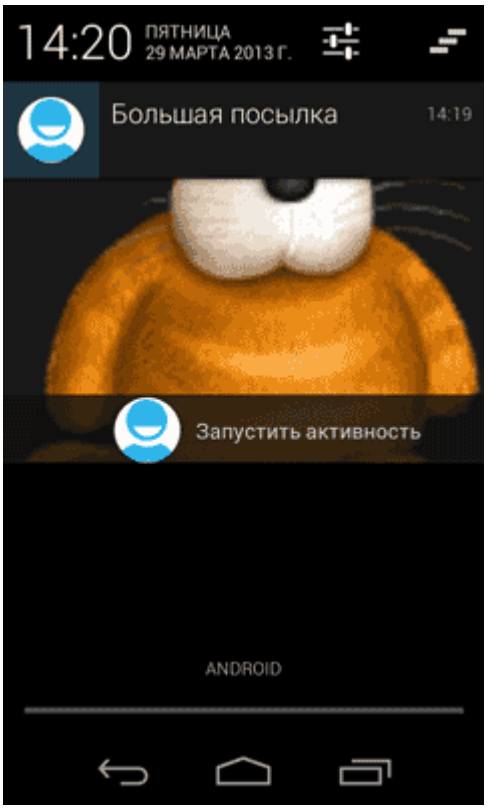

Рисунок Л.4.7 – Уведомление с большой картинкой

Существует ещё один стиль уведомлений – **InboxStyle**, напоминающий стиль писем в папке Входящие:

```
public void sendInboxStyleNotification(View view) {
    Intent intent = new Intent(this, SecondActivity.class);
    PendingIntent pIntent = PendingIntent.getActivity
                (this, 0, intent, 0);
    Builder builder = new Notification.Builder(this)
            .setTicker("Пришла посылка!")
            .setContentTitle("Уведомление в стиле Inbox")
            .setContentText("Inbox Style notification!!")
            .setSmallIcon(R.drawable.ic_launcher)
            .addAction(R.drawable.ic_launcher,
                    "Запустить активность", pIntent);

    Notification notification = new
          Notification.InboxStyle(builder)
            .addLine("Первое сообщение").
            addLine("Второе сообщение")
            .addLine("Третье сообщение").
            addLine("Четвертое сообщение")
            .setSummaryText("+2 more").build();

    notification.flags |= Notification.FLAG_AUTO_CANCEL;
    NotificationManager notificationManager =
            (NotificationManager) getSystemService
            (NOTIFICATION_SERVICE);
    notificationManager.notify(3, notification);
}
```

Пример уведомления в стиле **InboxStyle** показан на рисунке Л.4.8.

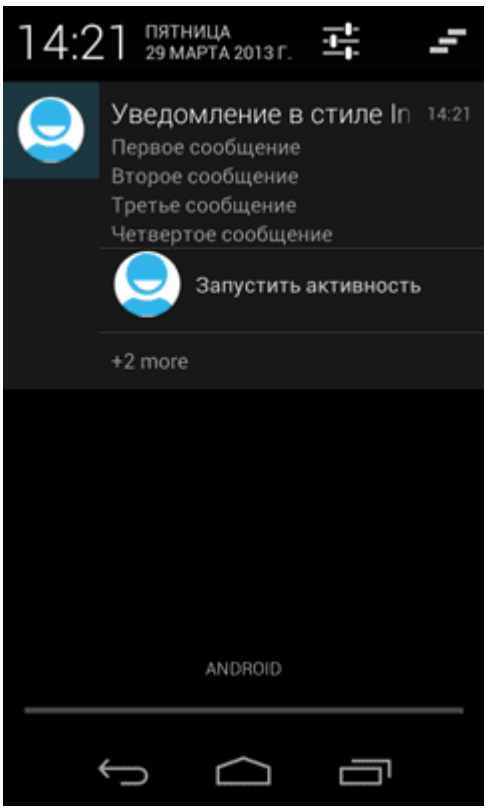

Рисунок Л.4.8 – Уведомление в стиле **InboxStyle**

Подводя итоги, следует отметить, у уведомлений очень много методов, которые можно использовать в своём приложении. Вот как может выглядеть полный набор:

```
new Notification.Builder(this.getApplicationContext())
        .setAutoCancel(boolean  autoCancel)
        .setContent(RemoteViews  views)
        .setContentInfo(CharSequence  info)
        .setContentIntent(PendingIntent  intent)
        .setContentText(CharSequence  text)
        .setContentTitle(CharSequence  title)
        .setDefaults(int  defaults)
        .setDeleteIntent(PendingIntent  intent))
        .setFullScreenIntent(PendingIntent  intent,
                boolean highPriority)
        .setLargeIcon(Bitmap  icon)
```

```
.setLights(int argb, int onMs, int offMs)
.setNumber(int number)
.setOngoing(boolean ongoing)
.setOnlyAlertOnce(boolean onlyAlertOnce)
.setPriority(int pri)
.setProgress(int max, int progress, boolean indeterminate)
.setShowWhen(boolean show)
.setSmallIcon(int icon, int level)
.setSmallIcon(int icon)
.setSound(Uri sound)
.setSound(Uri sound, int streamType)
.setStyle(Notification.Style style)
.setSubText(CharSequence text)
.setTicker(CharSequence tickerText, RemoteViews views)
.setTicker(CharSequence tickerText)
.setUsesChronometer(boolean b)
.setVibrate(long[] pattern)
.setWhen(long when)
.addAction(int icon, CharSequence title,
        PendingIntent intent)
.build()
```

3.5 Приоритеты уведомлений

Уведомления могут иметь разный приоритет.

В API 16 появился метод **setPriority()** со следующими константами (по мере увеличения):

- **Notification.PRIORITY_MIN**;
- **Notification.PRIORITY_LOW;**
- **Notification.PRIORITY_DEFAULT;**
- **Notification.PRIORITY_HIGH;**
- **Notification.PRIORITY_MAX**.

```
public void OnClick(View v) {
   final int NOTIFICATION_ID = 1;
```

```
    PendingIntent activityPendingIntent =
        getActivityPendingIntent();

    Notification notification = new
        NotificationCompat.Builder(this)
          .setContentTitle("Срочно!")
          .setContentText("Накорми кота!")
          .setSmallIcon(R.mipmap.ic_launcher)
          .setContentIntent(activityPendingIntent)
          .setPriority(Notification.PRIORITY_HIGH)
          .setDefaults(Notification.DEFAULT_ALL)
          .setCategory(Notification.CATEGORY_STATUS)
          .build();

    NotificationManagerCompat notificationManager =
        NotificationManagerCompat.from(this);
    notificationManager.notify(NOTIFICATION_ID,
        notification);
}

private PendingIntent getActivityPendingIntent() {
    Intent activityIntent = new Intent(this, MainActivity.class);
    activityIntent.addFlags
        (Intent.FLAG_ACTIVITY_SINGLE_TOP);
    return PendingIntent.getActivity(this, 0, activityIntent,
        PendingIntent.FLAG_UPDATE_CURRENT);
}
```

Чем выше приоритет уведомления, тем выше он находится среди остальных уведомлений. Таким образом, важные сообщения всегда будут наверху, даже если поступили позже других менее важных сообщений.

При этом следует учесть, что Google постоянно вносит какие-то изменения и добавления. Практически в каждой новой версии Android появляется что-что новое.

Вот пример изменений, которые произошли в API 23:

– удалили метод **setLatestEventInfo();**

– добавили новые методы **getLargeIcon()** и **getSmallIcon()**;

– добавили новое поле класса **CATEGORY_REMINDER** и объявили устаревшими поля **icon** и **largeIcon**.

4 Контрольные вопросы

1. Какие классы используются для создания уведомлений?
2. Назовите известные Вам настройки уведомлений.
3. Как создать уведомление, не запускающее никакую акивность?
4. Какие существуют способы настройки звука для уведомления?
5. Как настроить вибрацию при поступлении уведомления?
6. Что такое настойчивое уведомление? Как его создать?
7. Какие существуют способы настройки значков для уведомлений?
8. Как настроить уведомление с длинным текстом?
9. Какие приоритеты можно назначить уведомлению?

5 Задание для самостоятельного выполнения

Реализуйте приложение для создания уведомлений различного типа: с кнопками, в длинным текстом, с большой картинкой, в стиле **Inbox**. Настройки уведомлений (световая индикация, звук, вибрация и т. п.) задайте по желанию.

Лабораторная работа №5. Работа с графикой

1 Цель работы

Цель работы – получить практические навыки работы с 2D-графикой при проектировании мобильных кросслпатформенных приложений.

2 Содержание отчета

- титульный лист;
- краткое теоретическое описание;
- задание на лабораторную работу, включающее математическую формулировку задачи;
- результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений;

3 Порядок выполнения работы

Создайте новый проект **SimplePaint**. Далее в нём описываем новый класс **Draw2D**, который будет наследоваться от **View**. Именно в этом классе мы и будем проводить графические опыты. Щёлкаем правой кнопкой мыши на имени пакета и выбираем в меню **New | Java Class**. В открывшемся диалоговом окне устанавливаем имя для класса **Draw2D**.

Добавляем следующий код:

```
package ru.username.simplepaint; // у вас будет свой пакет

public class Draw2D extends View{
   public Draw2D(Context context) {
      super(context);
   }

   @Override
   protected void onDraw(Canvas canvas){
```

```
        super.onDraw(canvas);
    }
}
```

В данном коде мы наследуемся от класса **android.view.View** и переопределяем метод класса **onDraw(Canvas canvas)**.

Далее необходимо загрузить созданный класс при старте программы. Открываем основной файл активности **MainActivity** и заменяем строчку после **super.onCreate (savedInstanceState)**:

```
// эта строчка нам не нужна
// setContentView(R.layout.activity_main);
Draw2D draw2D = new Draw2D(this);
setContentView(draw2D);
```

То есть здесь мы сообщаем системе, что не нужно загружать разметку в экран активности. Вместо неё мы загрузим свой класс, у которого есть холст для рисования.

Подготовительные работы закончены. Перейдём к графике. Весь дальнейший код мы будем писать в классе **Draw2D**.

Для графики в мобильных приложенияъ используется холст **Canvas** – некая графическая поверхность для рисования. Прежде чем что-то рисовать, нужно определить некоторые параметры – цвет, толщина, фигура. Представьте себе, что вы рисуете на бумаге и в вашем распоряжении есть цветные карандаши, фломастеры, кисть, циркуль, ластик и т.п. Например, вы берёте толстый красный фломастер и рисуете жирную линию, затем берёте циркуль с жёлтым карандашом и рисуете окружность.

Вся работа с графикой происходит в методе **onDraw()** класса **Draw2D**. Создадим виртуальную кисть в классе. В методе укажем, что будем закрашивать всю поверхность белым цветом:

```
private Paint mPaint = new Paint();

@Override
protected void onDraw(Canvas canvas) {
   super.onDraw(canvas);

   // стиль Заливка
   mPaint.setStyle(Paint.Style.FILL);

   // закрашиваем холст белым цветом
   mPaint.setColor(Color.WHITE);
   canvas.drawPaint(mPaint);
}
```

Итак, холст готов. Далее начинается собственно рисование. Следуя описанному выше принципу, задаём перед каждым рисованием свои настройки и вызываем нужный метод. Например, для того, чтобы нарисовать жёлтый круг, мы включаем режим сглаживания, устанавливаем жёлтый цвет и вызываем метод **drawCircle()** с нужными координатами и заливаем окружность выбранным цветом. Получилось солнышко.

```
// Рисуем желтый круг
mPaint.setAntiAlias(true);
mPaint.setColor(Color.YELLOW);
canvas.drawCircle(950, 30, 25, mPaint);
```

Всегда соблюдайте очерёдность рисования. Если вы поместите данный код до заливки холста белым цветом, то ничего не увидите. Получится, как будто вы сначала нарисовали на стене солнце, а потом заклеили рисунок обоями.

Для рисования лужайки (зеленого прямоугольника) мы также задаём координаты и цвет:

```
// Рисуем зелёный прямоугольник
```

```
mPaint.setColor(Color.GREEN);
canvas.drawRect(20, 650, 950, 680, mPaint);
```

Далее выведем текст поверх лужайки. Устанавливаем синий цвет, стиль заливки, режим сглаживания и размер прямоугольника, в который будет вписан текст:

```
// Рисуем текст
mPaint.setColor(Color.BLUE);
mPaint.setStyle(Paint.Style.FILL);
mPaint.setAntiAlias(true);
mPaint.setTextSize(32);
canvas.drawText("Лужайка только для котов", 30, 648,
        mPaint);
```

При желании можно вывести текст под углом. Пусть это будет лучик солнца:

```
// Текст под углом
int x = 810;
int y = 190;

mPaint.setColor(Color.GRAY);
mPaint.setTextSize(27);
String str2rotate = "Лучик солнца!";

// Создаем ограничивающий прямоугольник для наклонного
// текста и поворачиваем холст по центру текста
canvas.rotate(-45, x + mRect.exactCenterX(),
            y + mRect.exactCenterY());

// Рисуем текст
mPaint.setStyle(Paint.Style.FILL);
canvas.drawText(str2rotate, x, y, mPaint);

// восстанавливаем холст
```

```
canvas.restore();
```

Завершаем композицию выводом рисунка из ресурсов:

```
// Выводим изображение
canvas.drawBitmap(mBitmap, 450, 530, mPaint);
```

В данном примере мы вручную подбирали размеры и координаты фигур для своего экрана. В реальных приложениях необходимо сначала вычислить размеры экрана у пользователя, а потом уже выводить фигуры в соответствии с полученными результатами. Иначе получится так, что некоторые элементы композиции просто не попадут на экран при вращении устройства. Допустим, в альбомном режиме вы установите у точки X значение 800, но в портретном режиме ширина экрана будет, скажем, 480, и точка окажется вне поле зрения. Поэтому следует позаботиться о вычислениях размеров экрана. Ниже представлен немного переделанный вариант для общего понимания.

Финальный рисунок выглядит следующим образом в двух ориентациях (рисунок Л.5.1-Л.5.2). При желании можно доработать приложение, например, уменьшив размеры кота и т. д.

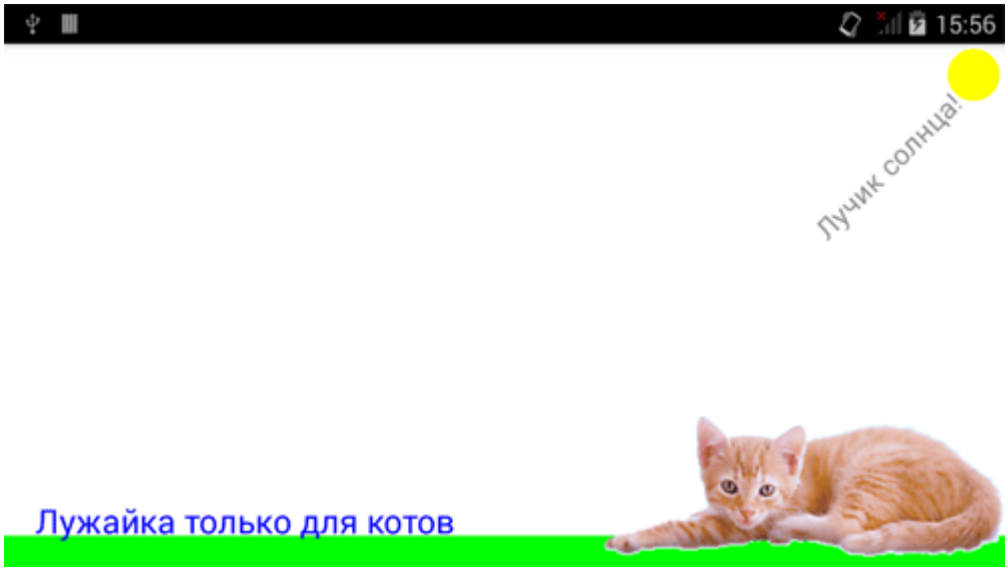

Рисунок Л.5.1 – Результат (альбомная ориентация)

Рисунок Л.5.1 – Результат (портретная ориентация)

Исходный код класса будет выглядеть следующим образом:

```
package ru.username.simplepaint;

import android.content.Context;
import android.content.res.Resources;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Rect;
import android.view.View;

public class Draw2D extends View {
```

```
private Paint mPaint = new Paint();
private Rect mRect = new Rect();
private Bitmap mBitmap;

public Draw2D(Context context) {
   super(context);

   // Выводим значок из ресурсов
   Resources res = this.getResources();
   mBitmap = BitmapFactory.decodeResource(res,
            R.drawable.cat_bottom);
}

@Override
protected void onDraw(Canvas canvas) {
   super.onDraw(canvas);

   int width = canvas.getWidth();
   int height = canvas.getHeight();

   // стиль Заливка
   mPaint.setStyle(Paint.Style.FILL);

   // закрашиваем холст белым цветом
   mPaint.setColor(Color.WHITE);
   canvas.drawPaint(mPaint);

   // Рисуем желтый круг
   mPaint.setAntiAlias(true);
   mPaint.setColor(Color.YELLOW);
   // canvas.drawCircle(950, 30, 25, mPaint);
   canvas.drawCircle(width - 30, 30, 25, mPaint);

   // Рисуем зеленый прямоугольник
   mPaint.setColor(Color.GREEN);
```

```
//  canvas.drawRect(20,  650, 950, 680, mPaint);
canvas.drawRect(0,  canvas.getHeight()  - 30, width, height,
        mPaint);

// Рисуем  текст
mPaint.setColor(Color.BLUE);
mPaint.setStyle(Paint.Style.FILL);
mPaint.setAntiAlias(true);
mPaint.setTextSize(32);
//  canvas.drawText("Лужайка  только для котов", 30,
       648, mPaint);
canvas.drawText("Лужайка  только для котов", 30,
       height  - 32, mPaint);

// Текст  под углом
// int x = 810;
int x = width - 170;
int y = 190;

mPaint.setColor(Color.GRAY);
mPaint.setTextSize(27);
String beam = "Лучик  солнца!";

canvas.save();
// Создаем ограничивающий  прямоугольник  для
// наклонного текста
// поворачиваем  холст по центру  текста
canvas.rotate(-45,  x + mRect.exactCenterX(),
        y + mRect.exactCenterY());

// Рисуем  текст
mPaint.setStyle(Paint.Style.FILL);
canvas.drawText(beam,  x, y,  mPaint);

// восстанавливаем  холст
```

```
        canvas.restore();

        // Выводим изображение
        // canvas.drawBitmap(mBitmap,  450, 530, mPaint);
        canvas.drawBitmap(mBitmap,
                width - mBitmap.getWidth(),
                height - mBitmap.getHeight()  - 10, mPaint);
    }
}
```

4 Контрольные вопросы

1. Как осуществляется вывод графики в мобильных приложениях?
2. Что такое холст? Для каких целей он служит?
3. В каком методе необходимо прописывать вывод графики?
4. Перечислите известные вам методы рисования. Какие параметры в них используются?
5. Как вывести на экран рисунок из ресурсов?
6. Как обеспечить корректный вывод графики для различных ориентаций устройства?

5 Задание для самостоятельного выполнения

Разработайте приложение с выводом на экран мобильного устройства звездного неба. Графическое изображение должно содержать: несколько звезд, месяц в виде серпа, космический корабль (его изображение можно загрузить из ресурсов).

Лабораторная работа №6. Создание анимации

1 Цель работы

Цель работы – ознакомиться и получить практические навыки использования методов создания анимации для мобильных приложений.

2 Содержание отчета

- титульный лист;
- краткое теоретическое описание;
- задание на лабораторную работу, включающее математическую формулировку задачи;
- результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений;

3 Порядок выполнения работы

Статичного контента не всегда бывает достаточно при проектировании приложений. Гораздо интереснее выглядят различные анимационные эффекты, которые привлекут внимание пользователя.

В Android доступны несколько видов анимации. Рассмотрим один из них для ознакомления.

В данной работе мы будем использовать анимацию из фигур, создав иллюзию восхода солнца. Также добавим анимацию аналоговых часов.

Создадим новый проект под названием "Sunrise" (Восход солнца).

3.1 Создание заготовок для анимации

Сначала нарисуем солнце. Создадим новую папку **drawable** в папке **res** (если такой папки нет). Далее в созданной папке создадим новый файл **sun.xml** следующего содержания:

```
<?xml version="1.0" encoding="utf-8"?>
<shape
xmlns:android="http://schemas.android.com/apk/res/android"
    android:dither="true"
    android:shape="oval" >

    <gradient
        android:endColor="#ffff6600"
        android:gradientRadius="150"
        android:startColor="#ffffcc00"
        android:type="radial"
        android:useLevel="false" />

    <size
        android:height="150dp"
        android:width="150dp" />

</shape>
```

Для изображения солнца мы использовали фигуру **Овал** с одинаковыми размерами, чтобы получить «солнечный круг». Чтобы рисунок солнца получился красивым, применим к нему градиент (плавное изменение цвета) от тёмно-жёлтого к светло-жёлтому.

Далее нарисуем «небо вокруг». В той же папке **drawable** создадим новый файл **sky.xml** следующего содержания:

```
<?xml version="1.0" encoding="utf-8"?>
<shape
xmlns:android="http://schemas.android.com/apk/res/android"
    android:dither="true"
    android:shape="rectangle" >

    <gradient
        android:angle="90"
```

```
        android:endColor="#ff000033"
        android:startColor="#ff0000ff" />

    </shape>
```

Мы задали фигуру в виде прямоугольника с голубым градиентом от нижнего края к верхнему.

Следующим этапом является рисование травы. Создаём в той же папке файл **grass.xml**:

```
    <?xml version="1.0" encoding="utf-8"?>
    <shape
xmlns:android="http://schemas.android.com/apk/res/android"
        android:dither="true"
        android:shape="rectangle" >

        <gradient
            android:angle="90"
            android:endColor="#ff003300"
            android:startColor="#ff009900" />

    </shape>
```

Трава будет нарисована в виде зелёного прямоугольника с градиентом.

Далее необходимо собрать фигуры вместе. Для начала откроем файл **strings.xml** в папке **res/values** и добавим несколько строковых ресурсов:

```
    <string name="sun">Солнце</string>
    <string name="grass">Трава</string>
    <string name="sky">Небо</string>
    <string name="clock">Часы</string>
    <string name="hour">Стрелка</string>
```

Откроем разметку главной активности **activity_main.xml** и добавим в неё несколько элементов **ImageView**:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <ImageView
       android:id="@+id/sky"
       android:layout_width="fill_parent"
       android:layout_height="fill_parent"
       android:contentDescription="@string/sky"
       android:src="@drawable/sky" />

    <ImageView
       android:id="@+id/sun"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_centerHorizontal="true"
       android:contentDescription="@string/sun"
       android:scaleType="fitCenter"
       android:src="@drawable/sun" />

    <ImageView
       android:id="@+id/grass"
       android:layout_width="fill_parent"
       android:layout_height="150dp"
       android:layout_alignParentBottom="true"
```

```
        android:contentDescription="@string/grass"
        android:src="@drawable/grass"  />

</RelativeLayout>
```

У всех элементов **ImageView** в атрибуте **android:src** мы прописали созданные фигуры, которые теперь можно видеть на экране (рисунок Л.6.1).

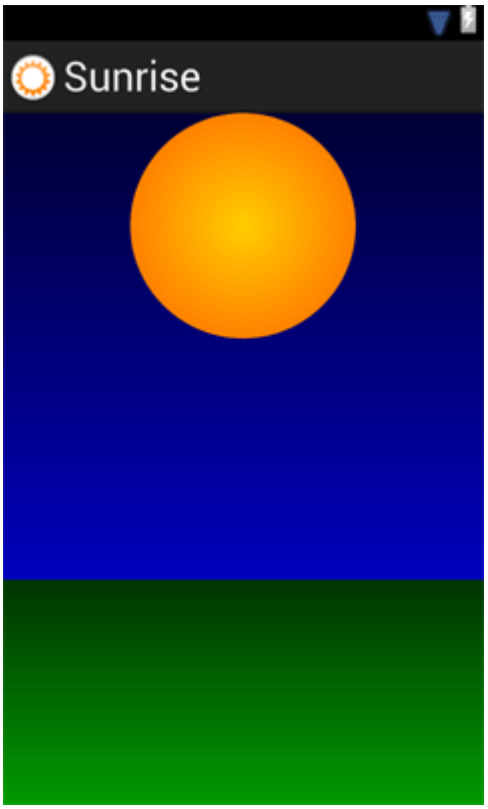

Рисунок Л.6.1 – Заготовка для анимации

3.2 Анимация восхода

В данном случае необходимо, чтобы солнце поднималось в верхнюю часть экрана. Создадим новую папку **res/anim**, в которой будут находиться файлы анимации.

Создадим в созданной папке новый файл **sun_rise.xml**:

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

```
    <set
xmlns:android="http://schemas.android.com/apk/res/android"
        android:duration="5000"
        android:fillAfter="true"
        android:interpolator=
                "@android:anim/accelerate_decelerate_interpolator"
        android:shareInterpolator="false" >

        <scale
           android:fromXScale="1.0"
           android:fromYScale="1.0"
           android:pivotX="50%"
           android:pivotY="50%"
           android:toXScale="1.5"
           android:toYScale="1.5"  />

        <translate
           android:fromYDelta="80%p"
           android:toYDelta="10%p"  />

        <alpha
           android:fromAlpha="0.3"
           android:toAlpha="1.0"  />

    </set>
```

В блоке **set** мы установили детали анимации. Например, параметр **android:duration** показывает, что анимация должна совершиться в течение 5 секунд. Параметр **fillAfter** управляет состоянием анимации – она не должна возвращаться в начало. Параметр **android:interpolator** использует системную константу для небольшого ускорения от начала к середине анимации и торможения от середины к концу анимации.

Внутри блока **set** устанавливаются специальные блоки, отвечающие за характер анимации: изменение размеров, позиции и прозрачности.

Например, фигура солнца по нашей задумке будет увеличиваться от своего изначального размера в полтора раза, раздуваясь равномерно от своей середины (scale).

Элемент **translate** двигает солнце по экрану вертикально вверх. Мы отталкиваемся относительно родительского элемента, используя суффикс "p". Солнце начинает движение в позиции 80% от родительского элемента по оси Y и заканчивает движение в позиции 10%.

При движении также меняется прозрачность солнца от полной прозрачности до полной непрозрачности (alpha).

Переходим непосредственно к программированию:

```
package ru.username.sunrise;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;

public class MainActivity extends Activity {

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

            // Получим ссылку на солнце
            ImageView sunImageView = (ImageView)
                        findViewById(R.id.sun);
            // Анимация для восхода солнца
```

```
            Animation sunRiseAnimation = AnimationUtils.
                    loadAnimation(this, R.anim.sun_rise);
            // Подключаем анимацию к нужному View
            sunImageView.startAnimation(sunRiseAnimation);
        }
    }
```

Запускаем проект и любуемся восходом солнца.

3.3 Анимация часов

Добавим к проекту часы с анимацией. Создадим в папке **res/drawable** файл **clock.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list
xmlns:android="http://schemas.android.com/apk/res/android" >

    <item>
        <shape
            android:dither="true"
            android:shape="oval" >
            <gradient
                android:endColor="#ffffffff"
                android:gradientRadius="100"
                android:startColor="#66ffffff"
                android:type="radial"
                android:useLevel="false" />

            <size
                android:height="100dp"
                android:width="100dp" />

            <stroke
                android:width="2dp"
```

```
            android:color="#99000000" />
        </shape>
    </item>
    <item
        android:bottom="44dp"
        android:left="48dp"
        android:right="48dp"
        android:top="5dp">
        <shape android:shape="rectangle" >
            <solid android:color="#99000000" />
        </shape>
    </item>

</layer-list>
```

Создадим в папке **res/anim** файл **clock_turn.xml** для анимации часов:

```
<?xml version="1.0" encoding="utf-8"?>
<set
xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="5000"
    android:fillAfter="true"
    android:interpolator="@android:anim/linear_interpolator"
    android:shareInterpolator="false" >

    <rotate
        android:fromDegrees="0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toDegrees="720" />

</set>
```

В анимации мы указали значение 720 градусов, чтобы часы сделали полный оборот два раза. Хотя вращается вся фигура, для пользователя будет казаться, что вращается только стрелка.

Добавим в разметку новый ImageView для часов:

```
<ImageView
   android:id="@+id/clock"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_alignParentBottom="true"
   android:layout_alignParentRight="true"
   android:contentDescription="@string/clock"
   android:padding="10dp"
   android:src="@drawable/clock" />
```

Теперь необходимо добавить код для анимации часов:

```
// Получим ссылку на часы
ImageView clockImageView = (ImageView)
            findViewById(R.id.clock);
// анимация для вращения часов
Animation clockTurnAnimation = AnimationUtils.
            loadAnimation(this, R.anim.clock_turn);
clockImageView.startAnimation(clockTurnAnimation);
```

Запустите проект, чтобы проверить, что всё работает.

Сейчас у часов одна минутная стрелка. Добавим ещё часовую стрелку. Создаём файл **hour_hand.xml** в папке **res/drawable**:

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list
xmlns:android="http://schemas.android.com/apk/res/android" >

   <item>
```

```
        <shape
            android:dither="true"
            android:shape="oval" >
            <solid android:color="#00000000" />

            <size
                android:height="100dp"
                android:width="100dp" />
        </shape>
    </item>
    <item
        android:bottom="44dp"
        android:left="48dp"
        android:right="48dp"
        android:top="15dp">
        <shape android:shape="rectangle" >
            <solid android:color="#99000000" />
        </shape>
    </item>

</layer-list>
```

Основные отличия от предыдущего файла – прозрачный круг и более короткая стрелка. При наложении на часы с минутной стрелкой, мы увидим часовую стрелку, а прозрачный круг мешать не будет.

Снова добавляем **ImageView** в разметку для часовой стрелки:

```
<ImageView
    android:id="@+id/hour_hand"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
```

```
    android:contentDescription="@string/clock"
    android:padding="10dp"
    android:src="@drawable/hour_hand" />
```

Компонент должен находиться в той же позиции, что и часы.

Создаём анимационный файл **hour_turn.xml** в папке **res/anim**:

```
<?xml version="1.0" encoding="utf-8"?>
<set
xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="5000"
    android:fillAfter="true"
    android:interpolator="@android:anim/linear_interpolator"
    android:shareInterpolator="false" >

    <rotate
        android:fromDegrees="180"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toDegrees="240" />

</set>
```

Начальная позиция установлена в значении 180 градусов, что соответствует 6 часам. При анимации стрелка повернётся на 60 градусов и будет соответствовать 8 часам. За это время минутная стрелка сделает два полных оборота, что соответствует двум часам (8-6).

Добавим анимацию в код:

```
// получим ссылку на часовую стрелку
ImageView hourImageView = (ImageView)
            findViewById(R.id.hour_hand);
// анимация для стрелки
```

```
Animation hourTurnAnimation = AnimationUtils.
            loadAnimation(this, R.anim.hour_turn);
// присоединяем анимацию
hourImageView.startAnimation(hourTurnAnimation);
```

Запускаем проект и наблюдаем за анимацией (рисунок Л.6.2). Как добавить кота, описано в предыдущей работе.

Рисунок Л.6.2 – Результат работы программы

4 Контрольные вопросы

1. Как нарисовать геометрическую фигуру с помощью ресурса?
2. В какой папке ресурсов должны находиться файлы анимации?
3. Как задать длительность анимации?
4. Какие параметры анимации можно настраивать внутри set-блока файла анимации? Как это делается?

5. Как применить созданную анимацию к элементу управления в активности приложения?

5 Задание для самостоятельного выполнения

Добавьте в приложение предыдущей лабораторной работы анимацию: звезды должны мерцать, месяц (луна) постепенно менять свои фазы, космический корабль – двигаться по определенной траектории. Поместите в приложение анимированные аналоговые часы, описанные в данной работе. В произвольный момент вращения стрелок (какой именно – выберите сами) на несколько секунд должна появляться кукушка.

Лабораторная работа №7. Работа с приложением Карты Google

1 Цель работы

Получить практические навыки использования Google-карт для поиска объектов в мобильных приложениях.

2 Содержание отчета

– наименование и цель работы;
– задание на лабораторную работу;
– текст программы;
– результаты работы программы.

3 Порядок выполнения работы

На большинстве стандартных телефонов и планшетов имеется программа **Карты Google**. Мы можем из своего приложения запустить эту программу с различными настройками.

Если планируется тестирование примеров на эмуляторе, то необходимо использовать виртуальное устройство с поддержкой Google APIs.

3.1 Запуск приложения для работы с картами при помощи намерения

Приложение для работы с картами запускается стандартным способом через намерение. Допустим, имеется кнопка, и код для щелчка будет следующим:

```
public void onClick(View view) {
    String geoUriString = "geo:0,10?z=2";
    Uri geoUri = Uri.parse(geoUriString);
    Intent mapIntent = new Intent(Intent.ACTION_VIEW,
                geoUri);
    startActivity(mapIntent);
```

}

Приложение **Карты Google** понимает специальный формат:

```
geo:latitude,longitude
geo:latitude,longitude?z=zoom
```

Первая строчка принимает два параметра: широту и долготу. Вторая строчка дополнительно имеет параметр масштабирования от 2 до 23 (2 – четверть планеты, 23 – максимальное приближение, например, крыша дома).

Документация утверждает, что можно использовать зум с уровнем 1 (весь мир), и раньше это действительно работало. Но после одного из обновлений карт значение 1 стало вызывать крах программы. Поэтому в качестве минимального значения зума сейчас используют 2.

В нашем примере для широты использовалось значение 0 – это экватор.

Имея нужные данные и зная необходимый формат, мы можем сформировать задание на запуск приложения для работы с картами. Как правило, на Android-телефонах это программа Карты (Maps). Если на телефоне установлены и другие карты (Яндекс.Карты, iGo), то сначала появится диалоговое окно, где пользователю будет предложено выбрать в каком приложении должна быть показана заданная точка.

Пример задания широты и долготы:

```
String geoURI = "geo:55.869555,37.503964?z=15";
Uri geo = Uri.parse(geoURI);
Intent geoIntent = new Intent(Intent.ACTION_VIEW, geo);
startActivity(geoIntent);
```

Еще один пример:

```
String geoUri = String.format("geo:%s,%s?z=15",
                    Double.toString(lat), Double.toString(lng));
```

```
Intent geoIntent = new Intent(Intent.ACTION_VIEW,
                Uri.parse(geoUri));
startActivity(geoIntent);
```

Запустив приложение и нажав на кнопку, мы запустим приложение, отвечающее за карты, и окажемся в нужном месте где-то в районе Ховрино (Москва).

Впрочем, совсем не обязательно знать точные координаты местоположения объекта. Можно поступить проще. Существует еще один формат:

geo:0,0?q=my+street+address

В этом случае мы можем не указывать координаты, а просто попросить у карт найти такую-то точку. Предположим, мы решили найти Бельгию и почему-то задали запрос на французском языке:

```
String geoUriString = "geo:0,0?q=Belgium";
// остальной код без изменений
```

Карты достаточно «умны», чтобы понять запрос и показать территорию Бельгии. Можно задать запрос и на русском языке. Как видно, в этом случае мы передаём нулевые координаты и добавляем к ним запрос **?q=**.

Также можно указать и уровень масштабирования:

String geoUriString = "geo:0,0?q=Кот Д'Ивуар&z=8";

Еще один пример запроса – показать кофейни рядом с Парижем.

geo:0,0?q=Coffee Shops near Paris, France

Пример запроса – театр кошек в Москве:

```
String geoUriString =
                "geo:0,0?q=москва+театр+кошек&z=8";
```

3.2 Просмотр улиц (Google StreetView)

Помимо карт, у Google есть приложение **Просмотр улиц (StreetView)**, интегрированное в Карты. Запуск его не отличается от запуска карт, только используется другой формат:

```
public void onClick(View view) {
    String geoUriString =
            "google.streetview:cbll=59.939448,30.328264&
            cbp=1,99.56,,1,2.0&mz=19";
    Uri geoUri = Uri.parse(geoUriString);
    Intent map = new Intent(Intent.ACTION_VIEW, geoUri);
    startActivity(map);
}
```

По этим параметрам выведется набережная канала Грибоедова в Санкт-Петербурге (рисунок Л.7.1).

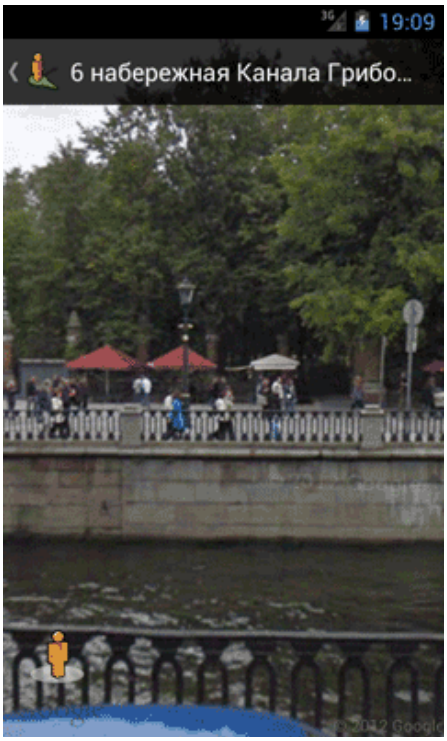

Рисунок Л.7.1 – Пример просмотра улиц

Сам формат выглядит следующим образом:

google.streetview:cbll=lat,lng&cbp=1,yaw,,pitch,
zoom&mz=mapZoom

Здесь:

- lat – широта;
- lng – долгота;
- yaw – центр панорамы в градусах по часовой стрелке с севера. Обязательно используйте две запятые;
- pitch – центр обзора панорамы в градусах от -90 (взор вверх) до 90 (взгляд вниз);
- zoom – масштаб панорамы. 1.0 = нормальный, 2.0 = приближение в 2 раза, 3.0 = в 4 раза и т. д.;
- mapZoom – масштабирование места карты, связанное с панорамой. Это значение используется при переходе на Карты.

Также можно внедрить карты в своё приложение. Но данная методика здесь не рассматривается.

4 Контрольные вопросы

1. Можно ли тестировать приложение, использующее **Карты Google**, на виртуальном устройстве?
2. Как запустить **Карты Google** с помощью намерений?
3. Каким образом можно задать масштабирование при просмотре карт?
4. Какие варианты запросов к картам можно использовать, помимо непосредственного задания широты и долготы?
5. Как запустить **Карты Google** в режиме просмотра улиц?

5 Заданий для самостоятельного выполнения

Разработайте приложение для поиска на Google-картах заданного места. Место должно задаваться либо адресом, либо широтой и долготой. Предусмотрите возможность «просмотра улиц» по сделанному запросу.

Список литературы

1. Павлова Е.А. Технологии разработки современных информационных систем на платформе Microsoft.NET [Электронный ресурс] : учебное пособие / Павлова Е.А. – Электрон. текстовые данные. – Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. – 128 с. – Режим доступа : http://www.iprbookshop.ru/89479.html

2. Ткаченко О. Н. Взаимодействие пользователей с интерфейсами информационных систем для мобильных устройств: исследование опыта [Электронный ресурс] : учебное пособие / Ткаченко О.Н. – Москва : Магистр : ИНФРА-М, 2020. – 152 с. – Текст : электронный. – Режим доступа: https://znanium.com/catalog/product/1045717

3. Куркин А.В. Программирование под платформу Andriod [Электронный ресурс] : учебное пособие / Куркин А.В.— Электрон. текстовые данные. – Санкт-Петербург : Университет ИТМО, 2015. – 35 с. – Режим доступа : http://www.iprbookshop.ru/67586.html.

4. Верескун, Д. М. Разработка мобильных приложений для бизнеса [Электронный ресурс] : учебное пособие / Д. М. Верескун. – Электрон. текстовые данные. – Саратов : Саратовский государственный технический университет имени Ю.А. Гагарина, ЭБС АСВ, 2012. – 51 с. – Режим доступа : http://www.iprbookshop.ru/76508.html

5. Программирование на языке Java [Электронный ресурс] : конспект лекций / А. В. Гаврилов, С. В. Клименков, А. Е. Харитонова, Е. А. Цопа. – Электрон. текстовые данные. — СПб. : Университет ИТМО, 2015. – 123 с. – Режим доступа : http://www.iprbookshop.ru/68692.html

6. Соколова В.В. Разработка мобильных приложений [Электронный ресурс] : учебное пособие/ Соколова В.В. – Электрон. текстовые данные. – Томск : Томский политехнический университет, 2014. – 176 с. – Режим доступа : http://www.iprbookshop.ru/34706.

Оглавление

Учебное издание

Иванова Елена Александровна
Крамаренко Татьяна Анатольевна

КРОССПЛАТФОРМЕННЫЕ ПРИЛОЖЕНИЯ

Практикум

В авторской редакции

Компьютерная верстка – Е. А. Иванова

Подписано в печать _____.2020. Формат 60 × 84 $^{1}/_{16}$.
Усл. печ. л. – 10. Уч.-изд. л. – 7,8.
Тираж __ экз. Заказ № .

Типография Кубанского государственного
аграрного университета.
350044, г. Краснодар, ул. Калинина, 13